目錄

简介

《Ceph 运维手册》汇总了 Ceph 在使用中常见的运维和操作问题，主要用于指导运维人员的相关工作。存储组的新员工，在对 Ceph 有了基础了解之后，也可以通过本手册进一步深入 Ceph 的使用和运维。

本书的内容大部分来自 Ceph 官方文档，另一部分来自技术博客，还有一部分来自实际使用中的经验总结。

环境

本手册是基于以下两种环境：

- Ubuntu 14.04， Ceph Hammer 版。
- CentOS 7.2， Ceph Jewel版。

作者

李海静
lihaijing@fiberhome.com

本书 GitBook 地址

点击下面的地址进行在线阅读：

https://lihaijing.gitbooks.io/ceph-handbook/content

本书 GitHub 地址

本书源文件托管在 GitHub 上，欢迎大家 Fork 本项目：

https://github.com/lihaijing/ceph-handbook

第一部分：常用操作

本部分介绍了 Ceph 集群的常用操作，包括进程的起停、集群的监控、用户管理、MON/OSD 的增加和删除、存储池的操作、修改集群的配置，以及 Crushmap 的管理、修改 Monitor 的 IP 等操作。

1. 操作集群

1.1 用 UPSTART 控制 CEPH

用 ceph-deploy 把 Ceph Cuttlefish 及更高版部署到 Ubuntu 14.04 上，你可以用基于事件的 Upstart 来启动、关闭 Ceph 节点上的守护进程。 Upstart 不要求你在配置文件里定义守护进程例程。

1.1.1 列出节点上所有的 Ceph 作业和实例

```
sudo initctl list | grep ceph
```

1.1.2 启动所有守护进程

要启动某一 Ceph 节点上的所有守护进程，用下列命令：

```
sudo start ceph-all
```

1.1.3 停止所有守护进程

要停止某一 Ceph 节点上的所有守护进程，用下列命令：

```
sudo stop ceph-all
```

1.1.4 按类型启动所有守护进程

要启动某一 Ceph 节点上的某一类守护进程，用下列命令：

```
sudo start ceph-osd-all
sudo start ceph-mon-all
sudo start ceph-mds-all
```

1.1.5 按类型停止所有守护进程

要停止某一 Ceph 节点上的某一类守护进程，用下列命令：

```
sudo stop ceph-osd-all
sudo stop ceph-mon-all
sudo stop ceph-mds-all
```

1.1.6 启动单个进程

要启动某节点上一个特定的守护进程例程，用下列命令之一：

```
sudo start ceph-osd id={id}
sudo start ceph-mon id={hostname}
sudo start ceph-mds id={hostname}
```

例如：

```
sudo start ceph-osd id=1
sudo start ceph-mon id=ceph-server
sudo start ceph-mds id=ceph-server
```

1.1.7 停止单个进程

要停止某节点上一个特定的守护进程例程，用下列命令之一：

```
sudo stop ceph-osd id={id}
sudo stop ceph-mon id={hostname}
sudo stop ceph-mds id={hostname}
```

例如：

```
sudo stop ceph-osd id=1
sudo stop ceph-mon id=ceph-server
sudo stop ceph-mds id=ceph-server
```

1.2 用 SYSTEMD 控制 CEPH

对于所有支持 `systemd` 的 Linux 发行版（CentOS 7, Fedora, Debian Jessie 8.x, SUSE），使用原生的 `systemd` 文件来代替传统的 `sysvinit` 脚本。不过需要注意，这和 Ceph 的版本也有关系。如果 CentOS 7 + Jewel，使用的就是 `systemd` 。

1.2.1 列出节点上所有的 Ceph systemd units

```
sudo systemctl status ceph\*.service ceph\*.target
```

1.2.2 启动所有守护进程

要启动某一 Ceph 节点上的所有守护进程，用下列命令：

```
sudo systemctl start ceph.target
```

1.2.3 停止所有守护进程

要停止某一 Ceph 节点上的所有守护进程，用下列命令：

```
sudo systemctl stop ceph\*.service ceph\*.target
```

1.2.4 按类型启动所有守护进程

要启动某一 Ceph 节点上的某一类守护进程，用下列命令：

```
sudo systemctl start ceph-osd.target
sudo systemctl start ceph-mon.target
sudo systemctl start ceph-mds.target
```

1.2.5 按类型停止所有守护进程

要停止某一 Ceph 节点上的某一类守护进程，用下列命令：

```
sudo systemctl stop ceph-mon\*.service ceph-mon.target
sudo systemctl stop ceph-osd\*.service ceph-osd.target
sudo systemctl stop ceph-mds\*.service ceph-mds.target
```

1.2.6 启动单个进程

要启动某节点上一个特定的守护进程例程，用下列命令之一：

```
sudo systemctl start ceph-osd@{id}
sudo systemctl start ceph-mon@{hostname}
sudo systemctl start ceph-mds@{hostname}
```

1.2.7 停止单个进程

要停止某节点上一个特定的守护进程例程，用下列命令之一：

```
sudo systemctl stop ceph-osd@{id}
sudo systemctl stop ceph-mon@{hostname}
sudo systemctl stop ceph-mds@{hostname}
```

1.3 把 CEPH 当服务运行

在某些环境下，还可以把 Ceph 当做服务来运行，比如 CentOS 7 + Hammer 。

1.3.1 启动所有守护进程

要启动本节点上的所有 Ceph 守护进程，用下列命令：

```
sudo service ceph [start|restart]
```

1.3.2 停止所有守护进程

要停止本节点上的所有 Ceph 守护进程，用下列命令：

```
sudo service ceph stop
```

1.3.3 按类型启动所有守护进程

要启动本节点上的某一类 Ceph 守护进程，用下列命令：

```
sudo service ceph start {daemon-type}
```

1.3.4 按类型停止所有守护进程

要停止本节点上的某一类 Ceph 守护进程，用下列命令：

```
sudo service ceph stop {daemon-type}
```

1.3.5 启动单个进程

要启动本节点上某个特定的守护进程例程，用下列命令：

```
sudo service ceph start {daemon-type}.{instance}
```

1.3.6 停止单个进程

要停止本节点上某个特定的守护进程例程，用下列命令：

```
sudo service ceph start {daemon-type}.{instance}
```

2. 监控集群

集群运行起来后，你可以用 `ceph` 工具来监控集群的状态，典型的监控项目包括检查 OSD 状态、monitor 的状态、PG 的状态和元数据服务器的状态（目前楚天云环境并没有部署元数据服务器）。

2.1 交互模式

要在交互模式下运行 `ceph` ，不要带参数运行 `ceph` ，例如：

```
ceph
ceph> health
ceph> status
ceph> quorum_status
ceph> mon_status
```

2.2 检查集群的监控状况

启动集群后、读写数据前，先检查下集群的健康状态。你可以用下面的命令检查：

```
ceph health
```

如果你的配置文件或 keyring 文件不在默认路径下，你得在命令中指定：

```
ceph -c /path/to/conf -k /path/to/keyring health
```

集群刚起来的时候，你也许会碰到像 `HEALTH_WARN XXX num placement groups stale` 这样的健康告警，等一会再检查下。集群准备好的话 `ceph health` 会给出 `HEALTH_OK` 这样的消息，这时候就可以开始使用集群了。

2.3 观察集群

要观察集群内正发生的事件，打开一个新终端，然后输入：

```
ceph -w
```

Ceph 会打印各种事件。例如一个包括 3 个 Mon、和 33 个 OSD 的 Ceph 集群可能会打印出这些：

```
  cluster b84b887e-9e0c-4211-8423-e0596939cd36
   health HEALTH_OK
   monmap e1: 3 mons at {OPS-ceph1=192.168.219.30:6789/0,OPS-cep
h2=192.168.219.31:6789/0,OPS-ceph3=192.168.219.32:6789/0}
          election epoch 94, quorum 0,1,2 OPS-ceph1,OPS-ceph2,OP
S-ceph3
   osdmap e1196: 33 osds: 33 up, 33 in
    pgmap v1789894: 2752 pgs, 7 pools, 590 GB data, 110 kobjects
          1154 GB used, 83564 GB / 84719 GB avail
              2752 active+clean
client io 0 B/s rd, 25852 B/s wr, 7 op/s

2016-11-04 20:20:13.682953 mon.0 [INF] pgmap v1789893: 2752 pgs:
 2752 active+clean; 590 GB data, 1154 GB used, 83564 GB / 84719
GB avail; 0 B/s rd, 44908 B/s wr, 14 op/s
2016-11-04 20:20:15.686275 mon.0 [INF] pgmap v1789894: 2752 pgs:
 2752 active+clean; 590 GB data, 1154 GB used, 83564 GB / 84719
GB avail; 0 B/s rd, 25852 B/s wr, 7 op/s
2016-11-04 20:20:16.690680 mon.0 [INF] pgmap v1789895: 2752 pgs:
 2752 active+clean; 590 GB data, 1154 GB used, 83564 GB / 84719
GB avail; 0 B/s rd, 32345 B/s wr, 16 op/s
2016-11-04 20:20:17.694259 mon.0 [INF] pgmap v1789896: 2752 pgs:
 2752 active+clean; 590 GB data, 1154 GB used, 83564 GB / 84719
GB avail; 0 B/s rd, 57170 B/s wr, 32 op/s
2016-11-04 20:20:18.698200 mon.0 [INF] pgmap v1789897: 2752 pgs:
 2752 active+clean; 590 GB data, 1154 GB used, 83564 GB / 84719
GB avail; 0 B/s rd, 33148 B/s wr, 16 op/s
2016-11-04 20:20:20.701697 mon.0 [INF] pgmap v1789898: 2752 pgs:
 2752 active+clean; 590 GB data, 1154 GB used, 83564 GB / 84719
GB avail; 0 B/s rd, 16333 B/s wr, 5 op/s
2016-11-04 20:20:21.705719 mon.0 [INF] pgmap v1789899: 2752 pgs:
 2752 active+clean; 590 GB data, 1154 GB used, 83564 GB / 84719
GB avail; 0 B/s rd, 17705 B/s wr, 12 op/s
```

输出信息里包含：

- 集群的 ID
- 集群健康状况
- monitor map 版本和 mon 法定人数状态

- OSD map 版本和 OSD 状态摘要
- PG map 版本
- PG 和 Pool 的数量
- 集群存储的数据量，对象的总量，以及集群的已用容量/总容量/可用容量
- 客户端的 iops 信息

2.4 检查集群的使用情况

要检查集群的数据用量及其在存储池内的分布情况，可以用 `df` 选项，它和 Linux 上的 `df` 相似。如下：

```
ceph df
```

得到的输出信息大致如下：

```
GLOBAL:
    SIZE       AVAIL      RAW USED     %RAW USED
    84719G     83564G        1154G          1.36
POOLS:
    NAME                  ID      USED         %USED      MAX AVAIL
    OBJECTS
    rbd                   0            0           0         41381G
          0
    volumes               1         284G        0.34         41381G
      57904
    images                2         224G        0.27         41381G
      39024
    backups               3            0           0         41381G
          1
    vms                   4       28736M        0.03         41381G
       4325
    volumes-ssd           5       53758M        0.06         41381G
      11854
    fitos_backup_pool     7        1286M           0         41381G
        354
```

输出的 **GLOBAL** 段展示了数据所占用集群存储空间的概要。

- **SIZE**： 集群的总容量。
- **AVAIL**： 集群的可用空间总量。
- **RAW USED**：已用存储空间总量。
- **% RAW USED**：已用存储空间比率。用此值对比 `full ratio` 和 `near full ratio` 来确保不会用尽集群空间。

输出的 **POOLS** 段展示了存储池列表及各存储池的大致使用率。本段没有反映出副本、克隆和快照的占用情况。例如，如果你把 1MB 的数据存储为对象，理论使用率将是 1MB ，但考虑到副本数、克隆数、和快照数，实际使用量可能是 2MB 或更多。

- **NAME**：存储池名字。
- **ID**：存储池唯一标识符。
- **USED**：大概数据量，单位为 KB 、MB 或 GB ；
- **%USED**：各存储池的大概使用率。
- **Objects**：各存储池内的大概对象数。

注意： **POOLS** 段内的数字是估计值，它们不包含副本、快照或克隆。因此，各 Pool 的 **USED** 和 **%USED** 数量之和不会达到 **GLOBAL** 段中的 **RAW USED** 和 **%RAW USED** 数量。

2.5 检查集群状态

要检查集群的状态，执行下面的命令：

```
ceph status
```

或者：

```
ceph -s
```

在交互模式下，输入 `status` 然后按回车：

```
ceph> status
```

Ceph 将打印集群状态，例如一个包括 1 个监视器、和 2 个 OSD 的小型 Ceph 集群可能打印：

```
cluster b370a29d-9287-4ca3-ab57-3d824f65e339
 health HEALTH_OK
 monmap e1: 1 mons at {ceph1=10.0.0.8:6789/0}, election epoch 2,
 quorum 0 ceph1
 osdmap e63: 2 osds: 2 up, 2 in
  pgmap v41332: 952 pgs, 20 pools, 17130 MB data, 2199 objects
        115 GB used, 167 GB / 297 GB avail
               1 active+clean+scrubbing+deep
              951 active+clean
```

2.6 检查 OSD 状态

你可以执行下列命令来确定 OSD 状态为 `up` 且 `in` ：

```
ceph osd stat
```

或者：

```
ceph osd dump
```

你也可以根据 OSD 在 CRUSH MAP 里的位置来查看：

```
ceph osd tree
```

Ceph 会打印 CRUSH 树，包括 host 的名称、它上面的 OSD 例程、状态及权重：

```
ID WEIGHT  TYPE NAME       UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 0.05997 root default
-2 0.01999     host ceph01
 0 0.01999         osd.0        up  1.00000          1.00000
-3 0.01999     host ceph02
 1 0.01999         osd.1        up  1.00000          1.00000
-4 0.01999     host ceph03
 2 0.01999         osd.2        up  1.00000          1.00000
```

另请参考本手册第一部分 3. 监控 OSD。

2.7 检查 Mon 状态

如果集群中有多个 Mon（很可能），你启动集群后、读写数据前应该检查 Mon 法定人数状态。运行着多个 Mon 时必须形成法定人数，最好周期性地检查 Mon 状态来确定它们在运行。

要查看 Mon map，执行下面的命令：

```
ceph mon stat
```

或者：

```
ceph mon dump
```

要检查监视器的法定人数状态，执行下面的命令：

```
ceph quorum_status -f json-pretty
```

Ceph 会返回法定人数状态，例如，包含 3 个监视器的 Ceph 集群可能返回下面的：

```
{
  "election_epoch": 94,
  "quorum": [
        0,
        1,
        2
  ],
  "quorum_names": [
        "OPS-ceph1",
        "OPS-ceph2",
        "OPS-ceph3"
  ],
  "quorum_leader_name": "OPS-ceph1",
  "monmap": { "epoch": 1,
      "fsid": "b84b887e-9e0c-4211-8423-e0596939cd36",
      "modified": "2016-11-04 20:19:57.333655",
      "created": "2016-06-23 14:53:07.171558",
      "mons": [
          {
              "rank": 0,
              "name": "OPS-ceph1",
              "addr": "192.168.219.30:6789\/0"
          },
          {
              "rank": 1,
              "name": "OPS-ceph2",
              "addr": "192.168.219.31:6789\/0"
          },
          {
              "rank": 2,
              "name": "OPS-ceph3",
              "addr":"192.168.219.32:6789\/0"
          }
        ]
   }
}
```

2.8 检查 MDS 状态

元数据服务器为 Ceph 文件系统提供元数据服务，不过在当前生产环境中并未部署 MDS 。元数据服务器有两种状态： `up | down` 和 `active | inactive` ，执行下面的命令查看元数据服务器状态为 `up` 且 `active` ：

```
ceph mds stat
```

要展示元数据集群的详细状态，执行下面的命令：

```
ceph mds dump
```

2.9 检查 PG 状态

PG 把对象映射到 OSD 。监控 PG 时，我们希望它们的状态是 `active` 且 `clean` 。详情请参考本手册第一部分 4. 监控 PG

2.10 使用管理套接字

Ceph 管理套接字允许你通过套接字接口查询守护进程，它们默认存在于 `/var/run/ceph` 下。要通过管理套接字访问某个守护进程，先登录它所在的主机、再执行下列命令：

```
ceph daemon {daemon-name}
ceph daemon {path-to-socket-file}
```

比如，这是下面这两种用法是等价的：

```
ceph daemon osd.0 foo
ceph daemon /var/run/ceph/ceph-osd.0.asok foo
```

用下列命令查看可用的管理套接字命令：

```
ceph daemon {daemon-name} help
```

管理套接字命令允许你在运行时查看和修改配置。

另外，你可以在运行时直接修改配置选项（也就是说管理套接字会绕过 Mon，不要求你直接登录宿主主机，不像 `ceph {daemon-type} tell {id} injectargs` 会依赖监视器）。

3. 监控 OSD

某 OSD 的状态可以是在集群内（ `in` ）或集群外（ `out` ）、也可以是运行着的（ `up` ）或不在运行的（ `down` ）。如果一个 OSD 处于 `up` 状态，它也可以是在集群之内 `in` （你可以读写数据）或者之外 `out` 。如果它以前是 `in` 但最近 `out` 了， Ceph 会把 PG 迁移到其他 OSD 上。如果某个 OSD `out` 了， CRUSH 就不会再分配 PG 给它。如果它 `down` 了，其状态也应该是 `out` 。默认在 OSD `down` 掉 300s 后会标记它为 `out` 状态。

注意：如果某个 **OSD** 状态为 **down & in** ，必定有问题，而且集群处于非健康状态。

OSD 监控的一个重要事情就是，当集群启动并运行时，所有 OSD 也应该是启动（ `up` ）并在集群内（ `in` ）运行的。用下列命令查看：

```
ceph osd stat
```

其结果会告诉你 osd map 的版本（ eNNNN ），总共有多少个 OSD 、几个是 `up` 的、几个是 `in` 的。

```
osdmap e26753: 3 osds: 2 up, 3 in
```

如果处于 `in` 状态的 OSD 多于 `up` 的，用下列命令看看哪些 `ceph-osd` 守护进程没在运行：

```
ceph osd tree ::

ID WEIGHT  TYPE NAME        UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 0.05997 root default
-2 0.01999     host ceph01
 0 0.01999         osd.0         up  1.00000          1.00000
-3 0.01999     host ceph02
 1 0.01999         osd.1         up  1.00000          1.00000
-4 0.01999     host ceph03
 2 0.01999         osd.2       down  1.00000          1.00000
```

如果有 OSD 处于 down 状态，请尝试启动该 OSD，启动命令见本手册第一部分 1. 操作集群。如果启动失败，请参考本手册第二部分 2. 常见 OSD 故障处理 中的相关部分进行处理。

4. 监控 PG

CRUSH 算法把 PG 分配到 OSD 时，它会根据存储池的副本数设置，把 PG 分配到不同的 OSD 上。比如，如果存储池设置为 3 副本， CRUSH 可能把它们分别分配到 `osd.1` 、 `osd.2` 、 `osd.3` 。考虑到 CRUSH Map 中设定的故障域，实际上 CRUSH 找出的是伪随机位置，所以在大型集群中，很少能看到 PG 被分配到了相邻的 OSD 。我们把涉及某个特定 PG 副本的一组 OSD 称为 **acting set** 。在某些情况下，位于 acting set 中的一个 OSD `down` 了或者不能为 PG 内的对象提供服务，这些情形发生时无需惊慌，常见原因如下：

- 你增加或移除了某个 OSD 。然后 CRUSH 算法把 PG 重新分配到了其他 OSD ，因此改变了 Acting Set 的构成，并且引发了 “backfill” 过程来进行数据迁移。
- 某个 OSD `down` 了、重启了，而现在正在恢复（ `recovering` ）。
- Acting Set 中的一个 OSD `down` 了，不能提供服务，另一个 OSD 临时接替其工作。

Ceph 靠 **Up Set** 处理客户端请求，它们是实际处理读写请求的 OSD 集合。大多数情况下 Up Set 和 Acting Set 是相同的。如果不同，说明可能 Ceph 正在迁移数据、某 OSD 在恢复、或者有别的问题。这种情况下， Ceph 通常表现为 “HEALTH WARN” 状态，还有 “stuck stale” 消息。

用下列命令获取 PG 列表：

```
ceph pg dump
```

查看指定 PG 的 Acting Set 或 Up Set 中包含的 OSD，执行：

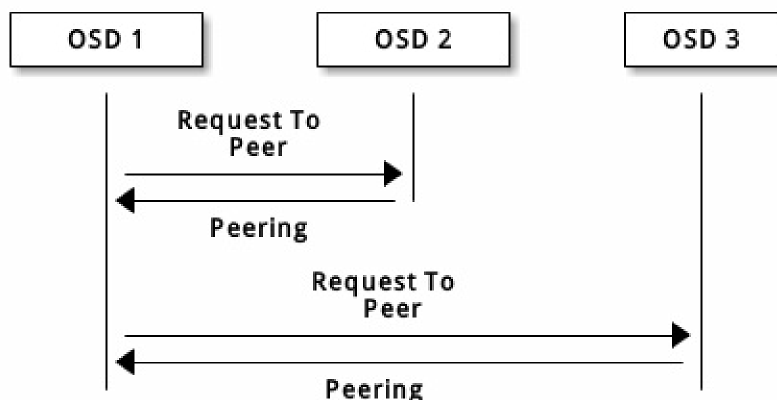
```
ceph pg map {pg-num}
```

命令的输出会告诉你 osdmap 版本（ eNNN ）、PG 号（ {pg-num} ）、Up Set 内的 OSD （ up[] ）、和 Acting Set 内的 OSD （ acting[] ）。

```
osdmap e1196 pg 0.2d (0.2d) -> up [13,30] acting [13,30]
```

4.1 节点互联

写入数据前，PG 必须处于 `active` 、而且应该是 `clean` 状态。假设某存储池的 PG 有 3 副本，为让 Ceph 确定 PG 的当前状态，PG 的主 OSD （即 acting set 内的第一个 OSD ）会与第二和第三 OSD 建立连接、并就 PG 的当前状态达成一致意见。



OSD 们也向 Mon 报告自己的状态。要排除节点互联的问题，请参考本手册第二部分 3. 常见 PG 故障处理 中的相关部分进行处理。

4.2 监控 PG 状态

如果你执行了 `ceph health` 、 `ceph -s` 、或 `ceph -w` 命令，你也许注意到了集群并非总返回 `HEALTH_OK` 。检查完 OSD 是否在运行后，你还应该检查 PG 的状态。你应该明白，在 PG 建立连接时集群不会返回 `HEALTH_OK` ：

- 刚刚创建了一个存储池，PG 还没达成一致。
- PG 正在恢复。
- 刚刚增加或删除了一个 OSD 。
- 刚刚修改了 CRUSH Map，PG 正在迁移。
- 某一 PG 的副本间的数据不一致。
- Ceph 正在洗刷一个 PG 的副本。
- Ceph 没有足够可用容量来完成回填操作。

如果是前述原因之一导致了 Ceph 返回 `HEALTH_WARN` ，无需紧张。很多情况下，集群会自行恢复；有些时候你得采取些措施。归置 PG 的一件重要事情是保证集群启动并运行着，所有 PG 都处于 `active` 状态、并且最好是 `clean` 状态。

用下列命令查看所有 PG 状态：

```
ceph pg stat
```

其结果会告诉你 PG map 的版本号（ vNNNNNN ）、PG 总数 x 、有多少 PG 处于某种特定状态，如 `active+clean` （ y ）。

```
vNNNNNN: x pgs: y active+clean; z MB data, aa MB used, bb MB / c
c MB avail
```

除了 PG 状态之外， Ceph 也会报告数据占据的空间（ aa ）、剩余可用空间（ bb ）和 PG 总容量。这些数字在某些情况下是很重要的：

- 集群快达到 `near full ratio` 或 `full ratio` 时。
- 由于 CRUSH 配置错误致使数据没能在集群内正确分布。

PG ID

PG IDs 由存储池号（不是存储池名字）、后面跟一个点（ . ）、再加一个 16 进制数字的 PG ID 。用 `ceph osd lspools` 可查看存储池号及其名字，例如，默认存储池 rbd 对应的存储池号是 0 。完整的 PG ID 格式如下：

```
{pool-num}.{pg-id}
```

典型例子：

```
0.1f
```

用下列命令获取 PG 列表：

```
ceph pg dump
```

你也可以让它输出到 JSON 格式，并保存到文件：

```
ceph pg dump -o {filename} --format=json
```

要查询某个 PG，用下列命令：

```
ceph pg {poolnum}.{pg-id} query
```

Ceph 会输出成 JSON 格式。

```
{
    "state": "active+clean",
    "snap_trimq": "[]",
    "epoch": 26760,
    "up": [
        1,
        2
    ],
    "acting": [
        1,
        2
    ],
    "actingbackfill": [
        "1",
        "2"
    ],
    "info": {
        "pgid": "0.2d",
        "last_update": "26708'96",
        "last_complete": "26708'96",
        "log_tail": "0'0",
        "last_user_version": 96,
        "last_backfill": "MAX",
        "purged_snaps": "[1~1]",
        "history": {
            "epoch_created": 1,
            "last_epoch_started": 26760,
            "last_epoch_clean": 26760,
    ......
    "recovery_state": [
        {
            "name": "Started\/Primary\/Active",
            "enter_time": "2016-11-05 11:01:12.719671",
```

```
            "might_have_unfound": [],
            "recovery_progress": {
                "backfill_targets": [],
                "waiting_on_backfill": [],
                "last_backfill_started": "0\/\/0\/\/-1",
                "backfill_info": {
                    "begin": "0\/\/0\/\/-1",
                    "end": "0\/\/0\/\/-1",
                    "objects": []
                },
                "peer_backfill_info": [],
                "backfills_in_flight": [],
                "recovering": [],
                "pg_backend": {
                    "pull_from_peer": [],
                    "pushing": []
                }
            },
            "scrub": {
                "scrubber.epoch_start": "26752",
                "scrubber.active": 0,
                "scrubber.waiting_on": 0,
                "scrubber.waiting_on_whom": []
            }
        },
        {
            "name": "Started",
            "enter_time": "2016-11-05 11:01:11.737126"
        }
    ],
    "agent_state": {}
}
```


4.3 找出故障 PG

如前所述，一个 PG 状态不是 `active+clean` 时未必有问题。一般来说，PG 卡住时 Ceph 的自修复功能可能会不起作用，卡住的状态细分为：

- **Unclean**: PG 里有些对象的副本数未达到期望值，它们应该进行恢复。

- **Inactive**: PG 不能处理读写请求，因为它们在等着一个持有最新数据的 OSD 回到 `up` 状态。
- **Stale**: PG 处于一种未知状态，因为存储它们的 OSD 有一阵子没向 Mon 报告了（由参数 `mon osd report timeout` 配置）。

为找出卡住的归置组，执行：

```
ceph pg dump_stuck [unclean|inactive|stale|undersized|degraded]
```

关于排除卡住的 PG 见请参考本手册第二部分 3. 常见 PG 故障处理 中的相关部分进行处理。

4.4 定位对象位置

要把对象数据存入 Ceph 对象存储， Ceph 客户端必须：

1. 设置对象名
2. 指定存储池

Ceph 客户端索取最新集群 map、并用 CRUSH 算法计算对象到 PG 的映射，然后计算如何动态地把 PG 分配到 OSD 。要定位对象位置，只需要知道对象名和存储池名字：

```
ceph osd map {poolname} {object-name}
```

练习：定位一个对象

我们先创建一个对象。给 rados put 命令指定一对象名、一个包含数据的测试文件路径、和一个存储池名字，例如：

```
rados put {object-name} {file-path} --pool=data
rados put test-object-1 testfile.txt --pool=data
```

用下列命令确认 Ceph 对象存储已经包含此对象：

```
rados -p data ls
```

现在可以定位对象了：

```
ceph osd map {pool-name} {object-name}
ceph osd map data test-object-1
```

Ceph 应该输出对象的位置，例如：

```
osdmap e537 pool 'data' (0) object 'test-object-1' -> pg 0.d1743
484 (0.4) -> up [1,0] acting [1,0]
```

要删除测试对象，用 rados rm 即可，如：

```
rados rm test-object-1 --pool=data
```

5. 用户管理

Ceph 把数据以对象的形式存于各存储池中。Ceph 用户必须具有访问存储池的权限才能够读写数据。另外，Ceph 用户必须具有执行权限才能够使用 Ceph 的管理命令。

5.1 授权（能力）

Ceph 用“能力”（ capabilities, caps ）这个术语来描述给认证用户的授权，这样才能使用 Mon、 OSD 和 MDS 的功能。能力也用于限制对某一存储池内的数据或某个命名空间的访问。 Ceph 管理员用户可在创建或更新普通用户时赋予他相应的能力。

能力的语法符合下面的形式：

```
{daemon-type} 'allow {capability}' [{daemon-type} 'allow {capability}']
```

Monitor 能力： Monitor 能力包括 `r` 、 `w` 、 `x` 和 `allow profile {cap}` ，例如：

```
mon 'allow rwx'
mon 'allow profile osd'
```

OSD 能力：OSD 能力包括 `r` 、 `w` 、 `x` 、 `class-read` 、 `class-write` 和 `profile osd` 。另外， OSD 能力还支持存储池和命名空间的配置。

```
osd 'allow {capability}' [pool={poolname}] [namespace={namespace-name}]
```

MDS 能力： MDS 能力比较简单，只需要 allow 或者空白，也不会解析更多选项。

```
mds 'allow'
```

注意： Ceph 对象网关守护进程（ `radosgw` ）是 Ceph 存储集群的一种客户端，所以它没被表示成一种独立的 Ceph 存储集群守护进程类型。

下面描述了各种能力。

`allow`

描述: 在守护进程的访问设置之前，仅对 MDS 隐含 `rw` 。

`r`

描述: 授予用户读权限，对 monitor 具有读权限才能获取 CRUSH map。

`w`

描述: 授予用户写对象的权限。

`x`

描述: 授予用户调用类方法的能力，即同时有读和写，且能在 monitor 上执行 `auth` 操作。

`class-read`

描述: 授予用户调用类读取方法的能力， `x` 的子集。

`class-write`

描述: 授予用户调用类写入方法的能力， `x` 的子集。

`*`

描述: 授权此用户读、写和执行某守护进程/存储池，且允许执行管理命令。

`profile osd`

描述: 授权一个用户以 OSD 身份连接其它 OSD 或 Monitor。授予 OSD 们允许其它 OSD 处理复制、心跳流量和状态报告。

`profile mds`

描述: 授权一个用户以 MDS 身份连接其它 MDS 或 Monitor。

`profile bootstrap-osd`

描述: 授权用户自举引导一个 OSD 。授予部署工具，像 `ceph-disk` 、 `ceph-deploy` 等等，这样它们在自举引导 OSD 时就有权限增加密钥了。

`profile bootstrap-mds`

描述: 授权用户自举引导一个 MDS。授予例如 `ceph-deploy` 的部署工具，这样它们在自举引导 MDS 时就有权限增加密钥了。

5.2 管理用户

用户管理功能可以让 Ceph 存储集群的管理员有能力去创建、更新和删除集群的普通用户。当创建或删除一个用户时，可能需要把 keys 分发给各客户端，以便它们可以加入到 keyring 文件中。

罗列用户

可以使用下面的命令罗列用户：

```
ceph auth list
```

Ceph 会列出集群中的所有用户。例如，在一个只有 2 个 OSD 的简单环境中， `ceph auth list` 将会输入类似下面的内容：

```
installed auth entries:

osd.0
        key: AQCvCbtToC6MDhAATtuT70Sl+DymPCfDSsyV4w==
        caps: [mon] allow profile osd
        caps: [osd] allow *
osd.1
        key: AQC4CbtTCFJBChAAVq5spj0ff4eHZICxIOVZeA==
        caps: [mon] allow profile osd
        caps: [osd] allow *
client.admin
        key: AQBHCbtT6APDHhAA5W00cBchwkQjh3dkKsyPjw==
        caps: [mds] allow
        caps: [mon] allow *
        caps: [osd] allow *
client.bootstrap-mds
        key: AQBICbtTOK9uGBAAdbe5zcIGHZL3T/u2g6EBww==
        caps: [mon] allow profile bootstrap-mds
client.bootstrap-osd
        key: AQBHCbtT4GxqORAADE5u7RkpCN/oo4e5W0uBtw==
        caps: [mon] allow profile bootstrap-osd
```

注意 `TYPE.ID` 这种用户表示方法，比如 `osd.0` 表示用户类型是 `osd` 且其 ID 是 `0` ， `client.admin` 表示用户类型是 `client` 且其 ID 是 `admin` （即默认的 `client.admin` 用户）。另外，每个用户条目都有一个 `key: <value>` 对，一个或多个 `caps:` 条目。

获取用户

获取某一特定用户的信息：

```
ceph auth get {TYPE.ID}
```

例如：

```
ceph auth get client.admin
```

还可以给命令加上 `-o {filename}` 选项把输入保存到一个文件中。

增加用户

增加一个用户就是创建一个用户名（即 `TYPE.ID` ）、一个密钥和任何包含在创建命令中的能力。有以下几种方式可以新增一个用户：

- `ceph auth add` ：此命令是最权威的新增用户方式。它回新建一个用户，产生一个 key ，并赋予用户任何给定的能力。
- `ceph auth get-or-create` ：这种方式通常是最便捷的一种，因为它的返回值是包含用户名（在方括号内）和密钥的格式。如果用户已经存在，该命令会返回密钥文件格式的用户名和密钥信息。可以使用 `-o {filename}` 选项把输入保存到一个文件中。
- `ceph auth get-or-create-key` ：该命令可以很方便地创建用户，但是只会返回用户的密钥。对于某些只需要密钥的用户（如 libvirt ）来说是很有用的。如果用户已经存在，该命令仅仅返回用户的密钥。可以使用 `-o {filename}` 选项把输入保存到一个文件中。

下面是几个例子：

```
ceph auth add client.john mon 'allow r' osd 'allow rw pool=liver
pool'
ceph auth get-or-create client.paul mon 'allow r' osd 'allow rw
pool=liverpool'
ceph auth get-or-create client.george mon 'allow r' osd 'allow r
w pool=liverpool' -o george.keyring
ceph auth get-or-create-key client.ringo mon 'allow r' osd 'allo
w rw pool=liverpool' -o ringo.key
```

修改用户的能力

`ceph auth caps` 命令允许你指定用户并改变用户的能力。设定新的能力会覆盖当前的能力。查看当前的能力可以使用 `ceph auth get USERTYPE.USERID` 命令。要增加能力，你应该在如下格式的命令中包含当前已经存在的能力：

```
ceph auth caps USERTYPE.USERID {daemon} 'allow [r|w|x|*|...] [po
ol={pool-name}] [namespace={namespace-name}]' [{daemon} 'allow [
r|w|x|*|...] [pool={pool-name}] [namespace={namespace-name}]']
```

例如：

```
ceph auth get client.john
ceph auth caps client.john mon 'allow r' osd 'allow rw pool=liverpool'
ceph auth caps client.paul mon 'allow rw' osd 'allow rwx pool=liverpool'
ceph auth caps client.brian-manager mon 'allow *' osd 'allow *'
```

要移除某个能力，你可能需要进行重置。如果你想取消某个用户对特定守护进程的所有访问权限，可以指定一个空的字符串。比如：

```
ceph auth caps client.ringo mon ' ' osd ' '
```

删除用户

想要删除一个用户，可以用 `ceph auth del` 命令：

```
ceph auth del {TYPE}.{ID}
```

其中， `{TYPE}` 是 `client` ， `osd` ， `mon` 或 `mds` 的其中一种。 `{ID}` 是用户的名字或守护进程的 ID 。

打印用户的密钥

打印某个用户的授权密钥到标准输出界面，可以执行如下命令：

```
ceph auth print-key {TYPE}.{ID}
```

其中， `{TYPE}` 是 `client` ， `osd` ， `mon` 或 `mds` 的其中一种。 `{ID}` 是用户的名字或守护进程的 ID 。

当需要往客户端软件中注入 Ceph 用户（如 libvirt）的密钥时，打印用户的密钥时很有用的。

```
mount -t ceph serverhost:/ mountpoint -o name=client.user,secret
=`ceph auth print-key client.user`
```

导入用户

使用 `ceph auth import` 命令并指定密钥文件，可以导入一个或多个用户：

```
ceph auth import -i /path/to/keyring
```

比如：

```
sudo ceph auth import -i /etc/ceph/ceph.keyring
```

5.3 密钥管理

当你通过客户端访问 Ceph 集群时，Ceph 客户端会使用本地的 keyring 文件。默认使用下列路径和名称的 keyring 文件：

- `/etc/ceph/$cluster.$name.keyring`
- `/etc/ceph/$cluster.keyring`
- `/etc/ceph/keyring`
- `/etc/ceph/keyring.bin`

你也可以在 `ceph.conf` 中另行指定 keyring 的路径，但不推荐这样做。

`$cluster` 元变量是你的 Ceph 集群名称，默认名称是 `ceph` 。`$name` 元变量是用户类型和 ID 。比如用户是 `client.admin` ，那就得到 `ceph.client.admin.keyring` 。

本小节介绍如何使用 `ceph-authtool` 工具来从客户端管理 keyring 。

创建密钥

创建一个空的 keyring 文件，使用 `--create-keyring` 或 `-C` 选项。比如：

```
ceph-authtool --create-keyring /path/to/keyring
```

创建一个包含多个用户的 keyring 文件，推荐使用 `$cluster.keyring` 作为文件名，并存放于 `/etc/ceph` 目录下。这样就无需在本地的 `ceph.conf` 文件中指定 keyring 的路径了。

```
sudo ceph-authtool -C /etc/ceph/ceph.keyring
```

创建仅包含一个用户的 keyring 文件时，建议使用集群名、用户类型和用户名称作为文件名，并存放于 `/etc/ceph` 目录下。

给密钥文件中增加用户

为了获取某个用户的 keyring 文件，可以使用 `ceph auth get` 命令加 `-o` 选项，以 keyring 文件格式来保存输出。比如：

```
sudo ceph auth get client.admin -o /etc/ceph/ceph.client.admin.k
eyring
```

当你想要向 keyring 文件中导入某个用户时，可以使用 ceph-authtool 来指定目的和源 keyring 文件。比如：

```
sudo ceph-authtool /etc/ceph/ceph.keyring --import-keyring /etc/
ceph/ceph.client.admin.keyring
```

创建用户

可以在 Ceph 客户端直接创建用户、密钥和能力。然后再导入 Ceph 集群。比如：

```
sudo ceph-authtool -n client.ringo --cap osd 'allow rwx' --cap m
on 'allow rwx' /etc/ceph/ceph.keyring
```

创建 keyring 文件、增加用户也可以同时进行。比如：

```
sudo ceph-authtool -C /etc/ceph/ceph.keyring -n client.ringo --c
ap osd 'allow rwx' --cap mon 'allow rwx' --gen-key
```

上述步骤中，新用户 client.ringo 仅存在于 keyring 文件中，还需要把新用户加入到 Ceph 集群中。

```
sudo ceph auth add client.ringo -i /etc/ceph/ceph.keyring
```

修改用户能力

修改记录在 keyring 文件中的用户能力时，需要指定 keyring 、用户和新的能力选项。比如：

```
sudo ceph-authtool /etc/ceph/ceph.keyring -n client.ringo --cap
osd 'allow rwx' --cap mon 'allow rwx'
```

更新 Ceph 存储集群的用户，你必须更新 keyring 文件中对应用户入口的信息。

```
sudo ceph auth import -i /etc/ceph/ceph.keyring
```

5.4 命令行使用方法

Ceph 支持通过下列方式使用用户名称和密钥。

`--id | --user`

描述：Ceph 通过类型和 ID 来确定用户，如 `TYPE.ID` 或 `client.admin`, `client.user1` 。使用 `--id` 或 `--user` 选项指定用户的 ID，比如，指定使用 `client.foo` 用户：

```
ceph --id foo --keyring /path/to/keyring health
ceph --user foo --keyring /path/to/keyring health
```

`--name | -n`

描述：使用 --name 或 -n 选项指定用户的全名（ TYPE.ID ），比如：

```
ceph --name client.foo --keyring /path/to/keyring health
ceph -n client.foo --keyring /path/to/keyring health
```

`--keyring`

描述：指定包含一个或多个用户及其密钥的 keyring 文件路径。 `--secret` 选项提供同样的功能，但对 Ceph RADOS Gateway 不生效（ `--secret` 选项有其他的作用）。比如：

```
sudo rbd map --id foo --keyring /path/to/keyring mypool/myimage
```

6. 增加/删除 Monitor

一个集群可以只有一个 monitor，我们推荐生产环境至少部署 3 个。 Ceph 使用 Paxos 算法的一个变种对各种 map 、以及其它对集群来说至关重要的信息达成共识。建议（但不是强制）部署奇数个 monitor 。Ceph 需要 mon 中的大多数在运行并能够互相通信，比如单个 mon，或 2 个中的 2 个，3 个中的 2 个，4 个中的 3 个等。初始部署时，建议部署 3 个 monitor。后续如果要增加，请一次增加 2 个。

6.1 增加 Monitor（手动）

1、在目标节点上，新建 mon 的默认目录。 `{mon-id}` 一般取为节点的 hostname 。

```
ssh {new-mon-host}
sudo mkdir /var/lib/ceph/mon/ceph-{mon-id}
```

2、创建一个临时目录（和第 1 步中的目录不同，添加 mon 完毕后需要删除该临时目录），来存放新增 mon 所需的各种文件，

```
mkdir {tmp}
```

3、获取 mon 的 keyring 文件，保存在临时目录下。

```
ceph auth get mon. -o {tmp}/{key-filename}
```

4、获取集群的 mon map 并保存到临时目录下。

```
ceph mon getmap -o {tmp}/{map-filename}
```

5、格式化在第 1 步中建立的 mon 数据目录。需要指定 mon map 文件的路径（获取法定人数的信息和集群的 `fsid` ）和 keyring 文件的路径。

```
sudo ceph-mon -i {mon-id} --mkfs --monmap {tmp}/{map-filename} --keyring {tmp}/{key-filename}
```

6、启动节点上的 mon 进程，它会自动加入集群。守护进程需要知道绑定到哪个 IP 地址，可以通过 `--public-addr {ip:port}` 选择指定，或在 `ceph.conf` 文件中进行配置 `mon addr` 。

```
ceph-mon -i {mon-id} --public-addr {ip:port}
```

6.2 增加 Monitor（ ceph-deploy ）

还可以通过 `ceph-deploy` 工具很方便地增加 MON。

1、登入 `ceph-deploy` 工具所在的 Ceph admin 节点，进入工作目录。

```
ssh {ceph-deploy-node}
cd /path/ceph-deploy-work-path
```

2、执行下列命令，新增 Monitor：

```
ceph-deploy mon create {host-name [host-name]...}
```

注意：在某一主机上新增 Mon 时，如果它不是由 `ceph-deploy new` 命令所定义的，那就必须把 `public network` 加入 `ceph.conf` 配置文件。

6.3 删除 Monitor（手动）

当你想要删除一个 mon 时，需要考虑删除后剩余的 mon 个数是否能够达到法定人数。

1、停止 mon 进程。

```
stop ceph-mon id={mon-id}
```

2、从集群中删除 monitor。

```
ceph mon remove {mon-id}
```

3、从 ceph.conf 中移除 mon 的入口部分（如果有）。

6.4 删除 Monitor（从不健康的集群中）

本小节介绍了如何从一个不健康的集群（比如集群中的 monitor 无法达成法定人数）中删除 `ceph-mon` 守护进程。

1、停止集群中所有的 `ceph-mon` 守护进程。

```
ssh {mon-host}
service ceph stop mon || stop ceph-mon-all
# and repeat for all mons
```

2、确认存活的 mon 并登录该节点。

```
ssh {mon-host}
```

3、提取 mon map。

```
ceph-mon -i {mon-id} --extract-monmap {map-path}
# in most cases, that's
ceph-mon -i `hostname` --extract-monmap /tmp/monmap
```

4、删除未存活或有问题的的 monitor。比如，有 3 个 monitors，`mon.a` 、`mon.b` 和 `mon.c` ，现在仅有 `mon.a` 存活，执行下列步骤：

```
monmaptool {map-path} --rm {mon-id}
# for example,
monmaptool /tmp/monmap --rm b
monmaptool /tmp/monmap --rm c
```

5、向存活的 monitor(s) 注入修改后的 mon map。比如，把 mon map 注入 `mon.a` ，执行下列步骤：

```
ceph-mon -i {mon-id} --inject-monmap {map-path}
# for example,
ceph-mon -i a --inject-monmap /tmp/monmap
```

6、启动存活的 monitor。

7、确认 monitor 是否达到法定人数（ `ceph -s` ）。

8、你可能需要把已删除的 monitor 的数据目录 `/var/lib/ceph/mon` 归档到一个安全的位置。或者，如果你确定剩下的 monitor 是健康的且数量足够，也可以直接删除数据目录。

6.5 删除 Monitor（ ceph-deploy ）

1、登入 `ceph-deploy` 工具所在的 Ceph admin 节点，进入工作目录。

```
ssh {ceph-deploy-node}
cd /path/ceph-deploy-work-path
```

2、如果你想删除集群中的某个 Mon ，可以用 `destroy` 选项。

```
ceph-deploy mon destroy {host-name [host-name]...}
```

注意： 确保你删除某个 Mon 后，其余 Mon 仍能达成一致。如果不可能，删除它之前可能需要先增加一个。

7. 增加/删除 OSD

如果您的集群已经在运行，你可以在运行时添加或删除 OSD 。

7.1 增加 OSD（手动）

要增加一个 OSD，要依次创建数据目录、把硬盘挂载到数据目录、把 OSD 加入集群、然后把它加入 CRUSH Map。

Tip： Ceph 喜欢统一的硬件，与存储池无关。如果你要新增容量不一的硬盘驱动器，还需调整它们的权重。但是，为实现最佳性能，CRUSH 的分级结构最好按类型、容量来组织。

1、创建 OSD。如果未指定 UUID， OSD 启动时会自动生成一个。下列命令会输出 OSD 号，后续步骤你会用到。

```
ceph osd create [{uuid} [{id}]]
```

如果指定了可选参数 {id} ，那么它将作为 OSD id 。要注意，如果此数字已使用，此命令会出错。

警告： 一般来说，我们不建议指定 {id} 。因为 ID 是按照数组分配的，跳过一些依然会浪费内存；尤其是跳过太多、或者集群很大时，会更明显。若未指定 {id} ，将用最小可用数字。

2、在新 OSD 主机上创建数据目录。

```
ssh {new-osd-host}
sudo mkdir /var/lib/ceph/osd/ceph-{osd-number}
```

3、如果准备用于 OSD 的是单独的磁盘而非系统盘，先把它挂载到刚创建的目录下：

```
ssh {new-osd-host}
sudo mkfs -t {fstype} /dev/{drive}
sudo mount -o user_xattr /dev/{hdd} /var/lib/ceph/osd/ceph-{osd-number}
```

4、初始化 OSD 数据目录。

```
ssh {new-osd-host}
ceph-osd -i {osd-num} --mkfs --mkkey
```

在启动 `ceph-osd` 前，数据目录必须是空的。

5、注册 OSD 认证密钥， `ceph-{osd-num}` 路径里的 `ceph` 值应该是 `$cluster-$id` ，如果你的集群名字不是 `ceph` ，那就用自己集群的名字。

```
ceph auth add osd.{osd-num} osd 'allow *' mon 'allow rwx' -i /var/lib/ceph/osd/ceph-{osd-num}/keyring
```

6、把新 OSD 加入 CRUSH Map 中，以便它可以开始接收数据。用 `ceph osd crush add` 命令把 OSD 加入 CRUSH 分级结构的合适位置。如果你指定了不止一个 bucket，此命令会把它加入你所指定的 bucket 中最具体的一个，并且把此 bucket 挪到你指定的其它 bucket 之内。

```
ceph osd crush add {id-or-name} {weight} [{bucket-type}={bucket-name} ...]
```

比如：

```
ceph osd crush add 21 0.08800 pool=ssd_root rack=ssd_rack01 host=ssd_ceph4
```

你也可以反编译 CRUSH Map、把 OSD 加入设备列表、以 bucket 的形式加入主机（如果它没在 CRUSH Map 里）、以条目形式把设备加入主机、分配权重、重编译并应用它，详情参见本手册第一部分 9. 修改 Crushmap 。

7、启动 OSD。把 OSD 加入 Ceph 后， OSD 就在配置里了。然而它还没运行，它现在的状态为 `down & out` 。你必须先启动 OSD 它才能收数据。在 Ubuntu 上执行：

```
sudo start ceph-osd id={osd-num}
```

一旦你启动了 OSD ，其状态就变成了 `up & in` 。

7.2 增加 OSD（ ceph-deploy ）

还可以通过 `ceph-deploy` 工具很方便的增加 OSD。

1、登入 `ceph-deploy` 工具所在的 Ceph admin 节点，进入工作目录。

```
ssh {ceph-deploy-node}
cd /path/ceph-deploy-work-path
```

2、列举磁盘。

执行下列命令列举一节点上的磁盘：

```
ceph-deploy disk list {node-name [node-name]...}
```

3、格式化磁盘。

用下列命令格式化（删除分区表）磁盘，以用于 Ceph ：

```
ceph-deploy disk zap {osd-server-name}:{disk-name}
ceph-deploy disk zap osdserver1:sdb
```

重要： 这会删除磁盘上的所有数据。

4、准备 OSD。

```
ceph-deploy osd prepare {node-name}:{data-disk}[:{journal-disk}]
ceph-deploy osd prepare osdserver1:sdb:/dev/ssd
ceph-deploy osd prepare osdserver1:sdc:/dev/ssd
```

`prepare` 命令只准备 OSD 。在大多数操作系统中，硬盘分区创建后，不用 `activate` 命令也会自动执行 `activate` 阶段（通过 Ceph 的 `udev` 规则）。

前例假定一个硬盘只会用于一个 OSD 守护进程，以及一个到 SSD 日志分区的路径。我们建议把日志存储于另外的驱动器以最优化性能；你也可以指定一单独的驱动器用于日志（也许比较昂贵）、或者把日志放到 OSD 数据盘（不建议，因为它有损性能）。前例中我们把日志存储于分好区的固态硬盘。

注意： 在一个节点运行多个 OSD 守护进程、且多个 OSD 守护进程共享一个日志分区时，你应该考虑整个节点的最小 CRUSH 故障域，因为如果这个 SSD 坏了，所有用其做日志的 OSD 守护进程也会失效。

5、准备好 OSD 后，可以用下列命令激活它。

```
ceph-deploy osd activate {node-name}:{data-disk-partition}[:{jou
rnal-disk-partition}]
ceph-deploy osd activate osdserver1:/dev/sdb1:/dev/ssd1
ceph-deploy osd activate osdserver1:/dev/sdc1:/dev/ssd2
```

`activate` 命令会让 OSD 进入 `up` 且 `in` 状态。该命令使用的分区路径是前面 `prepare` 命令创建的。

7.3 删除 OSD（手动）

要想缩减集群尺寸或替换硬件，可在运行时删除 OSD 。在 Ceph 里，一个 OSD 通常是一台主机上的一个 `ceph-osd` 守护进程、它运行在一个硬盘之上。如果一台主机上有多个数据盘，你得逐个删除其对应 `ceph-osd` 。通常，操作前应该检查集群容量，看是否快达到上限了，确保删除 OSD 后不会使集群达到 `near full` 比率。

警告： 删除 OSD 时不要让集群达到 `full ratio` 值，删除 OSD 可能导致集群达到或超过 `full ratio` 值。

1、停止需要剔除的 OSD 进程，让其他的 OSD 知道这个 OSD 不提供服务了。停止 OSD 后，状态变为 `down` 。

```
ssh {osd-host}
sudo stop ceph-osd id={osd-num}
```

2、将 OSD 标记为 `out` 状态，这个一步是告诉 mon，这个 OSD 已经不能服务了，需要在其他的 OSD 上进行数据的均衡和恢复了。

```
ceph osd out {osd-num}
```

执行完这一步后，会触发数据的恢复过程。此时应该等待数据恢复结束，集群恢复到 `HEALTH_OK` 状态，再进行下一步操作。

3、删除 CRUSH Map 中的对应 OSD 条目，它就不再接收数据了。你也可以反编译 CRUSH Map、删除 device 列表条目、删除对应的 host 桶条目或删除 host 桶（如果它在 CRUSH Map 里，而且你想删除主机），重编译 CRUSH Map 并应用它。详情参见本手册第一部分 9. 修改 Crushmap 。

```
ceph osd crush remove {name}
```

该步骤会触发数据的重新分布。等待数据重新分布结束，整个集群会恢复到 `HEALTH_OK` 状态。

4、删除 OSD 认证密钥：

```
ceph auth del osd.{osd-num}
```

5、删除 OSD 。

```
ceph osd rm {osd-num}
#for example
ceph osd rm 1
```

6、卸载 OSD 的挂载点。

```
sudo umount /var/lib/ceph/osd/$cluster-{osd-num}
```

7、登录到保存 `ceph.conf` 主拷贝的主机。

```
ssh {admin-host}
cd /etc/ceph
vim ceph.conf
```

8、从 `ceph.conf` 配置文件里删除对应条目。

```
[osd.1]
        host = {hostname}
```

9、从保存 `ceph.conf` 主拷贝的主机，把更新过的 `ceph.conf` 拷贝到集群其他主机的 `/etc/ceph` 目录下。

如果在 `ceph.conf` 中没有定义各 OSD 入口，就不必执行第 7 ~ 9 步。

8. 操作 Pool

如果你开始部署集群时没有创建存储池， Ceph 会用默认存储池 rbd 存放数据。存储池提供的功能：

- 自恢复力： 你可以设置在不丢数据的前提下允许多少 OSD 失效。对多副本存储池来说，此值是一对象应达到的副本数。典型配置是存储一个对象和它的一个副本（即 size = 2 ），但你可以更改副本数；对纠删编码的存储池来说，此值是编码块数（即纠删码配置里的 m = 2 ）。
- 归置组： 你可以设置一个存储池的 PG 数量。典型配置给每个 OSD 分配大约 100 个 PG，这样，不用过多计算资源就能得到较优的均衡。配置了多个存储池时，要考虑到这些存储池和整个集群的 PG 数量要合理。
- **CRUSH** 规则： 当你在存储池里存数据的时候，与此存储池相关联的 CRUSH 规则集可控制 CRUSH 算法，并以此操纵集群内对象及其副本的复制（或纠删码编码的存储池里的数据块）。你可以自定义存储池的 CRUSH 规则。
- 快照： 用 `ceph osd pool mksnap` 创建快照的时候，实际上创建了某一特定存储池的快照。

要把数据组织到存储池里，你可以列出、创建、删除存储池，也可以查看每个存储池的使用统计数据。

8.1 列出存储池

要列出集群的存储池，命令如下：

```
ceph osd lspools
```

在新安装好的集群上，默认只有一个 rbd 存储池。

8.2 创建存储池

创建存储池前可以先看看存储池、PG 和 CRUSH 配置参考。你最好在配置文件里重置默认 PG 数量，因为默认值并不理想。

例如：

```
osd pool default pg num = 100
osd pool default pgp num = 100
```

要创建一个存储池，执行：

```
ceph osd pool create {pool-name} {pg-num} [{pgp-num}] [replicate
d] \
        [crush-ruleset-name] [expected-num-objects]
ceph osd pool create {pool-name} {pg-num}  {pgp-num}   erasure \
        [erasure-code-profile] [crush-ruleset-name] [expected_nu
m_objects]
```

各参数含义如下：

- `{pool-name}` ：存储池名称，必须唯一。
- `{pg_num}` ： 存储池的 PG 数目。
- `{pgp_num}` ： 存储池的 PGP 数目，此值应该和 PG 数目相等。
- `{replicated|erasure}` ：存储池类型，可以是副本池（保存多份对象副本，以便从丢失的 OSD 恢复）或纠删池（获得类似 RAID5 的功能）。多副本存储池需更多原始存储空间，但已实现所有 Ceph 操作；纠删存储池所需原始存储空间较少，但目前仅实现了部分 Ceph 操作。
- `[crush-ruleset-name]` ：此存储池所用的 CRUSH 规则集名字。指定的规则集必须存在。对于多副本（ **replicated** ）存储池来说，其默认规则集由 `osd pool default crush replicated ruleset` 配置决定，此规则集必须存在。 对于用 `erasure-code` 编码的纠删码（ **erasure** ）存储池来说，不同的 `{pool-name}` 所使用的默认（ `default` ）纠删码配置是不同的，如果它不存在的话，会显式地创建它。
- `[erasure-code-profile=profile]` ：仅用于纠删存储池。指定纠删码配置文件，此配置必须已由 `osd erasure-code-profile set` 定义。
- `[expected-num-objects]` ：为这个存储池预估的对象数。设置此值（要同时把 **filestore merge threshold** 设置为负数）后，在创建存储池时就会拆分 PG 文件夹，以免运行时拆分文件夹导致延时增大。

关于如何计算合适的 `pg_num` 值，可以使用 Ceph 官方提供的一个计算工具 **pgcalc** 。

8.3 设置存储池配额

存储池配额可设置最大字节数、和/或每个存储池最大对象数。

```
ceph osd pool set-quota {pool-name} [max_objects {obj-count}] [max_bytes {bytes}]
```

例如：

```
ceph osd pool set-quota data max_objects 10000
```

要取消配额，设置为 `0` 即可。

8.4 删除存储池

要删除一存储池，执行：

```
ceph osd pool delete {pool-name} [{pool-name} --yes-i-really-really-mean-it]
```

如果你给自建的存储池创建了定制的规则集，你不需要存储池时最好也删除规则集。

```
ceph osd pool get {pool-name} crush_ruleset
```

加入规则集是 “123”，可以这样选择存储池：

```
ceph osd dump | grep "^pool" | grep "crush_ruleset 123"
```

如果你曾严格地创建了用户及其权限给一个存储池，但存储池已不存在，最好也删除那些用户。

```
ceph auth list | grep -C 5 {pool-name}
ceph auth del {user}
```

8.5 重命名存储池

要重命名一个存储池，执行：

```
ceph osd pool rename {current-pool-name} {new-pool-name}
```

如果重命名了一个存储池，且认证用户对每个存储池都有访问权限，那你必须用新存储池名字更新用户的能力（即 caps ）。

8.6 查看存储池统计信息

要查看某存储池的使用统计信息，执行命令：

```
rados df
```

8.7 给存储池做快照

要给某存储池做快照，执行命令：

```
ceph osd pool mksnap {pool-name} {snap-name}
```

8.8 删除存储池的快照

要删除某存储池的一个快照，执行命令：

```
ceph osd pool rmsnap {pool-name} {snap-name}
```

8.9 获取存储池选项值

要获取一个存储池的选项值，执行命令：

```
ceph osd pool get {pool-name} {key}
```

8.10 调整存储池选项值

要设置一个存储池的选项值，执行命令：

```
ceph osd pool set {pool-name} {key} {value}
```

常用选项介绍：

- `size` ：设置存储池中的对象副本数，详情参见设置对象副本数。仅适用于副本存储池。
- `min_size` ：设置 I/O 需要的最小副本数，详情参见设置对象副本数。仅适用于副本存储池。
- `pg_num` ：计算数据分布时的有效 PG 数。只能大于当前 PG 数。
- `pgp_num` ：计算数据分布时使用的有效 PGP 数量。小于等于存储池的 PG 数。
- `crush_ruleset` ：
- `hashpspool` ：给指定存储池设置/取消 HASHPSPOOL 标志。
- `target_max_bytes` ：达到 `max_bytes` 阀值时会触发 Ceph 冲洗或驱逐对象。
- `target_max_objects` ：达到 `max_objects` 阀值时会触发 Ceph 冲洗或驱逐对象。
- `scrub_min_interval` ：在负载低时，洗刷存储池的最小间隔秒数。如果是 0 ，就按照配置文件里的 osd_scrub_min_interval 。
- `scrub_max_interval` ：不管集群负载如何，都要洗刷存储池的最大间隔秒数。如果是 0 ，就按照配置文件里的 osd_scrub_max_interval 。
- `deep_scrub_interval` ："深度"洗刷存储池的间隔秒数。如果是 0 ，就按照配置文件里的 osd_deep_scrub_interval 。

8.11 设置对象副本数

要设置多副本存储池的对象副本数，执行命令：

```
ceph osd pool set {poolname} size {num-replicas}
```

重要： `{num-replicas}` 包括对象自身，如果你想要对象自身及其两份拷贝共计三份，指定 size 为 3 。

例如：

```
ceph osd pool set data size 3
```

你可以在每个存储池上执行这个命令。注意，一个处于降级模式的对象，其副本数小于 `pool size` ，但仍可接受 I/O 请求。为保证 I/O 正常，可用 `min_size` 选项为其设置个最低副本数。例如：

```
ceph osd pool set data min_size 2
```

这确保数据存储池里任何副本数小于 `min_size` 的对象都不会收到 I/O 了。

8.12 获取对象副本数

要获取对象副本数，执行命令：

```
ceph osd dump | grep 'replicated size'
```

Ceph 会列出存储池，且高亮 `replicated size` 属性。默认情况下， Ceph 会创建一对象的两个副本（一共三个副本，或 size 值为 3 ）。

9. 管理 Crushmap

CRUSH 算法通过计算数据存储位置来确定如何存储和检索。 CRUSH 授权 Ceph 客户端直接连接 OSD ，而非通过一个中央服务器或代理。数据存储、检索算法的使用，使 Ceph 避免了单点故障、性能瓶颈、和伸缩的物理限制。

CRUSH 需要一张集群的 Map，且使用 CRUSH Map 把数据伪随机地、尽量平均地分布到整个集群的 OSD 里。CRUSH Map 包含 OSD 列表、把设备汇聚为物理位置的“桶”列表、和指示 CRUSH 如何复制存储池里的数据的规则列表。

完全手动管理 CRUSH Map 也是可能的，在配置文件中设定：

```
osd crush update on start = false
```

9.1 编辑 CRUSH Map

要编辑现有的 CRUSH Map：

1. 获取 CRUSH Map；
2. 反编译 CRUSH 图；
3. 至少编辑一个设备、桶、规则；
4. 重编译 CRUSH Map；
5. 注入 CRUSH Map。

要激活 CRUSH Map 里某存储池的规则，找到通用规则集编号，然后把它指定到那个规则集。详情参见本手册第一部分 8. 操作 Pool 中调整存储池选项值部分。

获取 **CRUSH Map**

要获取集群的 CRUSH Map，执行命令：

```
ceph osd getcrushmap -o {compiled-crushmap-filename}
```

Ceph 将把 CRUSH 输出（ -o ）到你指定的文件，由于 CRUSH Map 是已编译的，所以编辑前必须先反编译。

反编译 CRUSH Map

要反编译 CRUSH Map，执行命令：

```
crushtool -d {compiled-crushmap-filename} -o {decompiled-crushmap-filename}
```

Ceph 将反编译（ -d ）二进制 CRUSH Map，且输出（ -o ）到你指定的文件。

编译 CRUSH Map

要编译 CRUSH Map，执行命令：

```
crushtool -c {decompiled-crush-map-filename} -o {compiled-crush-map-filename}
```

Ceph 将把已编译的 CRUSH Map 保存到你指定的文件。

注入 CRUSH Map

要把 CRUSH Map 应用到集群，执行命令：

```
ceph osd setcrushmap -i  {compiled-crushmap-filename}
```

Ceph 将把你指定的已编译 CRUSH Map 注入到集群。

9.2 CRUSH Map 参数

CRUSH Map 主要有 4 个段落。

1. 设备：由任意对象存储设备组成，即对应一个 `ceph-osd` 进程的存储器。Ceph 配置文件里的每个 OSD 都应该有一个设备。
2. 桶类型： 定义了 CRUSH 分级结构里要用的桶类型（ `types` ），桶由逐级汇聚的存储位置（如行、机柜、机箱、主机等等）及其权重组成。
3. 桶实例： 定义了桶类型后，还必须声明主机的桶类型、以及规划的其它故障域。
4. 规则： 由选择桶的方法组成。

CRUSH Map 之设备

为把 PG 映射到 OSD ， CRUSH Map 需要 OSD 列表（即配置文件所定义的 OSD 守护进程名称），所以它们首先出现在 CRUSH Map 里。要在 CRUSH Map 里声明一个设备，在设备列表后面新建一行，输入 `device` 、之后是唯一的数字 ID 、之后是相应的 `ceph-osd` 守护进程实例名字。

```
# devices
device {num} {osd.name}
```

例如：

```
# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3
```

CRUSH Map 之桶类型

CRUSH Map 里的第二个列表定义了 bucket （桶）类型，桶简化了节点和叶子层次。节点（或非叶子）桶在分级结构里一般表示物理位置，节点汇聚了其它节点或叶子，叶桶表示 `ceph-osd` 守护进程及其对应的存储媒体。

要往 CRUSH Map 中增加一种 bucket 类型，在现有桶类型列表下方新增一行，输入 `type` 、之后是惟一数字 ID 和一个桶名。按惯例，会有一个叶子桶为 `type 0` ，然而你可以指定任何名字（如 osd 、 disk 、 drive 、 storage 等等）：

```
# types
type {num} {bucket-name}
```

例如：

```
# types
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root
```

CRUSH Map 之桶层次

CRUSH 算法根据各设备的权重、大致统一的概率把数据对象分布到存储设备中。CRUSH 根据你定义的集群运行图分布对象及其副本， CRUSH Map 表达了可用存储设备以及包含它们的逻辑单元。

要把 PG 映射到跨故障域的 OSD ，一个 CRUSH Map 需定义一系列分级桶类型（即现有 CRUSH Map 的 `# type` 下）。创建桶分级结构的目的是按故障域隔离叶子节点，像主机、机箱、机柜、电力分配单元、机群、行、房间、和数据中心。除了表示叶子节点的 OSD ，其它分级结构都是任意的，你可以按需定义。

声明一个桶实例时，你必须指定其类型、惟一名称（字符串）、惟一负整数 ID（可选）、指定和各条目总容量/能力相关的权重、指定桶算法（通常是 straw ）、和哈希（通常为 0 ，表示哈希算法 rjenkins1 ）。一个桶可以包含一到多个条目，这些条目可以由节点桶或叶子组成，它们可以有个权重用来反映条目的相对权重。

你可以按下列语法声明一个节点桶：

```
[bucket-type] [bucket-name] {
        id [a unique negative numeric ID]
        weight [the relative capacity/capability of the item(s)]
        alg [the bucket type: uniform | list | tree | straw ]
        hash [the hash type: 0 by default]
        item [item-name] weight [weight]
}
```

例如，我们可以定义两个主机桶和一个机柜桶，机柜桶包含两个主机桶， OSD 被声明为主机桶内的条目：

```
host node1 {
        id -1
        alg straw
        hash 0
        item osd.0 weight 1.00
        item osd.1 weight 1.00
}

host node2 {
        id -2
        alg straw
        hash 0
        item osd.2 weight 1.00
        item osd.3 weight 1.00
}

rack rack1 {
        id -3
           alg straw
        hash 0
        item node1 weight 2.00
        item node2 weight 2.00
}
```

调整桶的权重

Ceph 用双精度类型数据表示桶权重。权重和设备容量不同，我们建议用 1.00 作为 1TB 存储设备的相对权重，这样 0.5 的权重大概代表 500GB 、 3.00 大概代表 3TB 。较高级桶的权重是所有叶子桶的权重之和。

一个桶的权重是一维的，你也可以计算条目权重来反映存储设备性能。例如，如果你有很多 1TB 的硬盘，其中一些数据传输速率相对低、其他的数据传输率相对高，即使它们容量相同，也应该设置不同的权重（如给吞吐量较低的硬盘设置权重 0.8 ，较高的设置 1.20 ）。

CRUSH Map 之规则

CRUSH Map 支持“ CRUSH 规则”的概念，用以确定一个存储池里数据的分布。CRUSH 规则定义了归置和复制策略、或分布策略，用它可以规定 CRUSH 如何放置对象副本。对大型集群来说，你可能创建很多存储池，且每个存储池都有它自己的 CRUSH 规则集和规则。默认的 CRUSH Map 里，每个存储池有一条规则、一个规则集被分配到每个默认存储池。

注意： 大多数情况下，你都不需要修改默认规则。新创建存储池的默认规则集是 `0` 。

规则格式如下：

```
rule <rulename> {

        ruleset <ruleset>
        type [ replicated | erasure ]
        min_size <min-size>
        max_size <max-size>
        step take <bucket-type>
        step [choose|chooseleaf] [firstn|indep] <N> <bucket-type
>
        step emit
}
```

参数说明：

- `ruleset` ：区分一条规则属于某个规则集的手段。给存储池设置规则集后激活。
- `type` ：规则类型，目前仅支持 `replicated` 和 `erasure` ，默认是 `replicated` 。
- `min_size` ：可以选择此规则的存储池最小副本数。
- `max_size` ：可以选择此规则的存储池最大副本数。
- `step take <bucket-name>` ：选取起始的桶名，并迭代到树底。
- `step choose firstn {num} type {bucket-type}` ：选取指定类型桶的数量，这个数字通常是存储池的副本数（即 pool size ）。如果 `{num} == 0` ，选择 `pool-num-replicas` 个桶（所有可用的）；如果 `{num} > 0 && < pool-num-replicas` ，就选择那么多的桶；如果 `{num} < 0` ，它意味着选择 `pool-num-replicas - {num}` 个桶。
- `step chooseleaf firstn {num} type {bucket-type}` ：选择 `{bucket-`

type}` 类型的桶集合，并从各桶的子树里选择一个叶子节点。桶集合的数量通常是存储池的副本数（即 pool size ）。如果 `{num} == 0` ，选择 `pool-num-replicas` 个桶（所有可用的）；如果 `{num} > 0 && < pool-num-replicas` ，就选择那么多的桶；如果 `{num} < 0` ，它意味着选择 `pool-num-replicas - {num}` 个桶。

- `step emit` ：输出当前值并清空堆栈。通常用于规则末尾，也适用于相同规则应用到不同树的情况。

9.3 主亲和性

某个 Ceph 客户端读写数据时，总是连接 acting set 里的主 OSD （如 `[2, 3, 4]` 中， `osd.2` 是主的）。有时候某个 OSD 与其它的相比并不适合做主 OSD （比如其硬盘慢、或控制器慢）。最大化硬件利用率时为防止性能瓶颈（特别是读操作），你可以调整 OSD 的主亲和性，这样 CRUSH 就尽量不把它用作 acting set 里的主 OSD 了。

```
ceph osd primary-affinity <osd-id> <weight>
```

主亲和性默认为 `1` （就是说此 OSD 可作为主 OSD ）。此值合法范围为 `0-1` ，其中 `0` 意为此 OSD 不能用作主的， `1` 意为 OSD 可用作主的。此权重 `< 1` 时， CRUSH 选择主 OSD 时选中它的可能性就较低。

9.4 增加/移动 OSD

要增加或移动在线集群里 OSD 所对应的 CRUSH Map 条目，执行 ceph osd crush set 命令。

```
ceph osd crush set {id-or-name} {weight} {bucket-type}={bucket-name} [{bucket-type}={bucket-name} ...]
```

9.5 调整 OSD 的 CRUSH 权重

要调整在线集群中某个 OSD 的 CRUSH 权重，执行命令：

```
ceph osd crush reweight {name} {weight}
```

9.6 删除 OSD

要从在线集群里把某个 OSD 彻底踢出 CRUSH Map，或仅踢出某个指定位置的 OSD，执行命令：

```
ceph osd crush remove {name} {<ancestor>}
```

9.7 增加桶

要在运行集群的 CRUSH Map 中新建一个桶，用 ceph osd crush add-bucket 命令：

```
ceph osd crush add-bucket {bucket-name} {bucket-type}
```

9.8 移动桶

要把一个桶移动到 CRUSH Map 里的不同位置，执行命令：

```
ceph osd crush move {bucket-name} {bucket-type}={bucket-name} [{
bucket-type}={bucket-name} ...]
```

9.9 删除桶

要把一个桶从 CRUSH Map 的分级结构中删除，可用此命令：

```
ceph osd crush remove {bucket-name}
```

注意：从 CRUSH 分级结构里删除时必须是空桶。

9.10 可调选项

从 v0.74 起，如果 CRUSH 可调选项不是最优值（ v0.73 版里的默认值） Ceph 就会发出健康告警，有两种方法可消除这些告警：

1、调整现有集群上的可调选项。注意，这可能会导致一些数据迁移（可能有 10% 之多）。这是推荐的办法，但是在生产集群上要注意此调整对性能带来的影响。此命令可启用较优可调选项：

```
ceph osd crush tunables optimal
```

如果切换得不太顺利（如负载太高）且切换才不久，或者有客户端兼容问题（较老的 cephfs 内核驱动或 rbd 客户端、或早于 bobtail 的 librados 客户端），你可以这样切回：

```
ceph osd crush tunables legacy
```

2、不对 CRUSH 做任何更改也能消除报警，把下列配置加入 `ceph.conf` 的 `[mon]` 段下：

```
mon warn on legacy crush tunables = false
```

为使变更生效需重启所有监视器，或者执行下列命令：

```
ceph tell mon.\* injectargs --no-mon-warn-on-legacy-crush-tunabl
es
```

9.11 CRUSH Map 实例

假设你想让大多数存储池映射到使用大容量硬盘的 OSD 上，但是其中一些存储池映射到使用高速 SSD 的 OSD 上。在同一个 CRUSH Map 内有多个独立的 CRUSH 层级结构是可能的，定义两棵树、分别有自己的根节点 —— 一个用于机械硬盘（如 root platter ）、一个用于 SSD （如 root ssd ），具体的 CRUSH Map 内容如下：

```
# devices
device 0 osd.0
```

```
device 1 osd.1
device 2 osd.2
device 3 osd.3
device 4 osd.4
device 5 osd.5
device 6 osd.6
device 7 osd.7

# types
type 0 osd
type 1 host
type 2 root

# buckets
host ceph-osd-ssd-server-1 {
        id -1
        alg straw
        hash 0
        item osd.0 weight 1.00
        item osd.1 weight 1.00
}

host ceph-osd-ssd-server-2 {
        id -2
        alg straw
        hash 0
        item osd.2 weight 1.00
        item osd.3 weight 1.00
}

host ceph-osd-platter-server-1 {
        id -3
        alg straw
        hash 0
        item osd.4 weight 1.00
        item osd.5 weight 1.00
}

host ceph-osd-platter-server-2 {
        id -4
```

```
        alg straw
        hash 0
        item osd.6 weight 1.00
        item osd.7 weight 1.00
}

root platter {
        id -5
        alg straw
        hash 0
        item ceph-osd-platter-server-1 weight 2.00
        item ceph-osd-platter-server-2 weight 2.00
}

root ssd {
        id -6
        alg straw
        hash 0
        item ceph-osd-ssd-server-1 weight 2.00
        item ceph-osd-ssd-server-2 weight 2.00
}

# rules
rule replicated_ruleset {
          ruleset 0
          type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}

rule platter {
        ruleset 1
        type replicated
        min_size 0
        max_size 10
        step take platter
        step chooseleaf firstn 0 type host
```

```
        step emit
}

rule ssd {
        ruleset 2
        type replicated
        min_size 0
        max_size 4
        step take ssd
        step chooseleaf firstn 0 type host
        step emit
}

rule ssd-primary {
        ruleset 3
        type replicated
        min_size 5
        max_size 10
        step take ssd
        step chooseleaf firstn 1 type host
        step emit
        step take platter
        step chooseleaf firstn -1 type host
        step emit
}
```

然后你可以设置一个存储池，让它使用 SSD 规则：

```
ceph osd pool set <poolname> crush_ruleset 2
```

同样，用 `ssd-primary` 规则将使存储池内的各归置组用 SSD 作主 OSD ，普通硬盘作副本。

10. 修改 MON IP

Ceph 客户端和其他 Ceph 守护进程通过 `ceph.conf` 来发现 monitor。但是 monitor 之间是通过 mon map 而非 `ceph.conf` 来发现彼此。

10.1 修改 MON IP（正确的方法）

仅修改 `ceoh.conf` 中 mon 的 IP 是不足以确保集群中的其他 monitor 收到更新的。要修改一个 mon 的 IP，你必须先新增一个使用新 IP 的 monitor（参考1.6 增加/删除 Monitor），确保这个新 mon 成功加入集群并形成法定人数。然后，删除使用旧 IP 的 mon。最后，更新 `ceph.conf` ，以便客户端和其他守护进程可以知道新 mon 的 IP。

比如，假设现有 3 个 monitors：

```
[mon.a]
        host = host01
        addr = 10.0.0.1:6789
[mon.b]
        host = host02
        addr = 10.0.0.2:6789
[mon.c]
        host = host03
        addr = 10.0.0.3:6789
```

把 `mon.c` 变更为 `mon.d` 。按照本手册第一部分 6. 增加/删除 Monitor，增加一个 `mon.d` ，host 设为 `host04` ，IP 地址设为 `10.0.0.4` 。先启动 `mon.d` ，再按照 6. 增加/删除 Monitor 删除 `mon.c` ，否则会破坏法定人数。

10.2 修改 MON IP（不太友好的方法）

有时，monitor 需要迁移到一个新的网络中、数据中心的其他位置或另一个数据中心。这时，需要为集群中所有的 monitors 生成一个新的 mon map （指定了新的 MON IP），再注入每一个 monitor 中。

还以前面的 mon 配置为例。假定想把 monitor 从 `10.0.0.x` 网段改为 `10.1.0.x` 网段，这两个网段直接是不通的。执行下列步骤：

1、获取 mon map。

```
ceph mon getmap -o {tmp}/{filename}
```

2、下面的例子说明了 monmap 的内容。

```
$ monmaptool --print {tmp}/{filename}

monmaptool: monmap file {tmp}/{filename}
epoch 1
fsid 224e376d-c5fe-4504-96bb-ea6332a19e61
last_changed 2012-12-17 02:46:41.591248
created 2012-12-17 02:46:41.591248
0: 10.0.0.1:6789/0 mon.a
1: 10.0.0.2:6789/0 mon.b
2: 10.0.0.3:6789/0 mon.c
```

3、删除已有的 monitors。

```
$ monmaptool --rm a --rm b --rm c {tmp}/{filename}

monmaptool: monmap file {tmp}/{filename}
monmaptool: removing a
monmaptool: removing b
monmaptool: removing c
monmaptool: writing epoch 1 to {tmp}/{filename} (0 monitors)
```

4、新增 monitor。

```
$ monmaptool --add a 10.1.0.1:6789 --add b 10.1.0.2:6789 --add c
 10.1.0.3:6789 {tmp}/{filename}

monmaptool: monmap file {tmp}/{filename}
monmaptool: writing epoch 1 to {tmp}/{filename} (3 monitors)
```

5、检查 monmap 的新内容。

```
$ monmaptool --print {tmp}/{filename}

monmaptool: monmap file {tmp}/{filename}
epoch 1
fsid 224e376d-c5fe-4504-96bb-ea6332a19e61
last_changed 2012-12-17 02:46:41.591248
created 2012-12-17 02:46:41.591248
0: 10.1.0.1:6789/0 mon.a
1: 10.1.0.2:6789/0 mon.b
2: 10.1.0.3:6789/0 mon.c
```

此时，我们假定 monitor 已在新位置安装完毕。下面的步骤就是分发新的 monmap 并注入到各新 monitor 中。

1、停止所有的 monitor 。必须停止 mon 守护进程才能进行 monmap 注入。

2、注入 monmap。

```
ceph-mon -i {mon-id} --inject-monmap {tmp}/{filename}
```

3、重启各 monitors 。

11. 修改集群配置

启动 Ceph 存储集群时，各守护进程都从同一个配置文件（即默认的 `ceph.conf` ）里查找它自己的配置。 `ceph.conf` 中可配置参数很多，有时我们需要根据实际环境对某些参数进行修改。

修改的方式分为两种：直接修改 `ceph.conf` 配置文件中的参数值，修改完后需要重启 Ceph 进程才能生效。或在运行中动态地进行参数调整，无需重启进程。

11.1 查看运行时配置

如果你的 Ceph 存储集群在运行，而你想看一个在运行进程的配置，用下面的命令：

```
ceph daemon {daemon-type}.{id} config show | less
```

如果你现在位于 osd.0 所在的主机，命令将是：

```
ceph daemon osd.0 config show | less
```

11.2 修改配置文件

Ceph 配置文件可用于配置存储集群内的所有守护进程、或者某一类型的所有守护进程。要配置一系列守护进程，这些配置必须位于能收到配置的段落之下，比如：

`[global]`

描述： `[global]` 下的配置影响 Ceph 集群里的所有守护进程。
实例： `auth supported = cephx`

`[osd]`

描述： `[osd]` 下的配置影响存储集群里的所有 `ceph-osd` 进程，并且会覆盖 `[global]` 下的同一选项。
实例： `osd journal size = 1000`

`[mon]`

描述： `[mon]` 下的配置影响集群里的所有 `ceph-mon` 进程，并且会覆盖 `[global]` 下的同一选项。
实例： `mon addr = 10.0.0.101:6789`

`[mds]`

描述： `[mds]` 下的配置影响集群里的所有 `ceph-mds` 进程，并且会覆盖 `[global]` 下的同一选项。
实例： `host = myserver01`

`[client]`

描述： `[client]` 下的配置影响所有客户端（如挂载的 Ceph 文件系统、挂载的块设备等等）。
实例： `log file = /var/log/ceph/radosgw.log`

全局设置影响集群内所有守护进程的例程，所以 `[global]` 可用于设置适用所有守护进程的选项。但可以用这些覆盖 `[global]` 设置：

1. 在 `[osd]` 、 `[mon]` 、 `[mds]` 下更改某一类进程的配置。
2. 更改特定进程的设置，如 `[osd.1]` 。

覆盖全局设置会影响所有子进程，明确剔除的例外。

11.3 运行中动态调整

Ceph 可以在运行时更改 `ceph-osd` 、 `ceph-mon` 、 `ceph-mds` 守护进程的配置，此功能在增加/降低日志输出、启用/禁用调试设置、甚至是运行时优化的时候非常有用。Ceph 集群提供两种方式的调整，使用 `tell` 的方式和 `daemon` 设置的方式。

tell 方式设置

下面是使用 tell 命令的修改方法：

```
ceph tell {daemon-type}.{id or *} injectargs --{name} {value} [-
-{name} {value}]
```

用 `osd` 、 `mon` 、 `mds` 中的一个替代 `{daemon-type}` ，你可以用星号（ `*` ）更改一类进程的所有实例配置、或者更改某一具体进程 ID （即数字或字母）的配置。例如提高名为 `osd.0` 的 `ceph-osd` 进程之调试级别的命令如下：

```
ceph tell osd.0 injectargs --debug-osd 20 --debug-ms 1
```

在 `ceph.conf` 文件里配置时用空格分隔关键词；但在命令行使用的时候要用下划线或连字符（ `_` 或 `-` ）分隔，例如 `debug osd` 变成 `debug-osd` 。

daemon 方式设置

除了上面的 tell 的方式调整，还可以使用 daemon 的方式进行设置。

1、获取当前的参数

```
ceph daemon osd.1 config get mon_osd_full_ratio

{
"mon_osd_full_ratio": "0.98"
}
```

2、修改配置

```
ceph daemon osd.1 config set mon_osd_full_ratio 0.97

{
"success": "mon_osd_full_ratio = '0.97' "
}
```

3、检查配置

```
ceph daemon osd.1 config get mon_osd_full_ratio

{
"mon_osd_full_ratio": "0.97"
}
```

注意：重启进程后配置会恢复到默认参数，在进行在线调整后，如果这个参数是后续是需要使用的，那么就需要将相关的参数写入到配置文件 `ceph.conf` 当中。

两种设置方式的使用场景

使用 tell 的方式适合对整个集群进行设置，使用 `*` 号进行匹配，就可以对整个集群的角色进行设置。而出现节点异常无法设置时候，只会在命令行当中进行报错，不太便于查找。

使用 daemon 进行设置的方式就是一个个的去设置，这样可以比较好的反馈，此方法是需要在设置的角色所在的主机上进行设置。

12. 日志和调试

一般来说，你应该在运行时增加调试选项来调试问题；也可以把调试选项添加到 Ceph 配置文件里来调试集群启动时的问题，然后查看 `/var/log/ceph` （默认位置）下的日志文件。

Tip： 调试输出会拖慢系统，这种延时有可能掩盖竞争条件。

日志记录是资源密集型任务。如果你碰到的问题在集群的某个特定区域，只启用那个区域对应的日志功能即可。例如，你的 OSD 运行良好、元数据服务器却有问题，这时应该先打开那个可疑元数据服务器实例的调试日志；如果不行再打开各子系统的日志。

重要： 详尽的日志每小时可能超过 1GB ，如果你的系统盘满了，这个节点就会停止工作。

如果你要打开或增加 Ceph 日志级别，确保有足够的系统盘空间。滚动日志文件的方法见下面的 加快日志更迭 小节。集群稳定运行后，可以关闭不必要的调试选项以优化运行。集群在运行中记录调试输出信息会拖慢系统、且浪费资源。

12.1 运行时

如果你想在运行时查看某一进程的配置，必须先登录对应主机，然后执行命令：

```
ceph daemon {daemon-name} config show | less
```

例如：

```
ceph daemon osd.0 config show | less
```

要在运行时激活 Ceph 的调试输出（即 `dout()` ），用 `ceph tell` 命令把参数注入运行时配置：

```
ceph tell {daemon-type}.{daemon id or *} injectargs --{name} {va
lue} [--{name} {value}]
```

用 `osd` 、 `mon` 或 `mds` 替代 `{daemon-type}` 。还可以用星号（ `*` ）把配置应用到同类型的所有守护进程，或者指定具体守护进程的 ID 。例如，要给名为 `ods.0` 的 `ceph-osd` 守护进程提高调试级别，用下列命令：

```
ceph tell osd.0 injectargs --debug-osd 0/5
```

`ceph tell` 命令会通过 monitor 起作用。如果你不能绑定 monitor，仍可以登录你要改的那台主机然后用 ceph daemon 来更改。例如：

```
sudo ceph daemon osd.0 config set debug_osd 0/5
```

12.2 启动时

要在启动时激活调试输出（即 `dout()` ），你得把选项加入配置文件。各进程共有配置可写在配置文件的 `[global]` 段下，某类进程的配置可写在对应的守护进程段下（如 `[mon]` 、 `[osd]` 、 `[mds]` ）。例如：

```
[global]
        debug ms = 1/5

[mon]
        debug mon = 20
        debug paxos = 1/5
        debug auth = 2

[osd]
        debug osd = 1/5
        debug filestore = 1/5
        debug journal = 1
        debug monc = 5/20

[mds]
        debug mds = 1
        debug mds balancer = 1
        debug mds log = 1
        debug mds migrator = 1
```

12.3 加快日志更迭

如果你的系统盘比较满，可以修改 `/etc/logrotate.d/ceph` 内的日志滚动配置以加快滚动。在滚动频率后增加一个日志 `size` 选项（达到此大小就滚动）来加快滚动（通过 cronjob ）。例如默认配置大致如此：

```
rotate 7
weekly
compress
sharedscripts
```

增加一个 `size` 选项。

```
rotate 7
weekly
size 500M
compress
sharedscripts
```

然后，打开 crontab 编辑器。

```
crontab -e
```

最后，增加一条用以检查 `/etc/logrorate.d/ceph` 文件的语句。

```
30 * * * * /usr/sbin/logrotate /etc/logrotate.d/ceph >/dev/null
2>&1
```

本例中每 30 分钟检查一次 `/etc/logrorate.d/ceph` 文件。

12.4 VALGRIND 工具

调试时可能还需要追踪内存和线程问题。你可以在 Valgrind 中运行单个守护进程、一类进程、或整个集群。 Valgrind 是计算密集型程序，应该只用于开发或调试 Ceph，否则它会拖慢系统。Valgrind 的消息会记录到 `stderr` 。

12.5 子系统、日志和调试选项

大多数情况下，你可以通过子系统打开调试日志输出。

CEPH 子系统概览

各子系统都有日志级别用于分别控制其输出日志和暂存日志（内存中），你可以分别为这些子系统设置不同的记录级别。 Ceph 的日志级别从 1 到 20 ， 1 是简洁、 20 是详尽。通常，内存驻留日志不会发送到输出日志，除非：

- 致命信号出现，或者
- 源码中的 `assert` 被触发，或者
- 明确要求发送。

调试选项允许用单个数字同时设置日志级别和内存级别，这会将二者设置为一样的值。比如，如果你指定 `debug ms = 5` ， Ceph 会把日志级别和内存级别都设置为 `5` 。也可以分别设置，第一个选项是日志级别、第二个是内存级别，二者必须用斜线（ / ）分隔。假如你想把 `ms` 子系统的调试日志级别设为 `1` 、内存级别设为 `5` ，可以写为 `debug ms = 1/5` ，如下：

```
debug {subsystem} = {log-level}/{memory-level}
#for example
debug mds log = 1/20
```

Ceph 子系统及其默认日志和内存级别具体见 SUBSYSTEM, LOG AND DEBUG SETTINGS 。一旦你完成调试，应该恢复默认值，或一个适合平常运营的级别。

日志记录选项

关于日志记录选项的详细信息，可以参考 Ceph 官方文档的对应内容 LOGGING SETTING 。

第二部分：故障处理

本部分就 Ceph 存储集群常见的问题做了归纳和总结，方便运维人员进行故障排除。

1. 常见 MON 故障处理

Monitor 维护着 Ceph 集群的信息，如果 Monitor 无法正常提供服务，那整个 Ceph 集群就不可访问。一般来说，在实际运行中，Ceph Monitor的个数是 2n + 1 (n >= 0) 个，在线上至少3个，只要正常的节点数 >= n+1，Ceph 的 Paxos 算法就能保证系统的正常运行。所以，当 Monitor 出现故障的时候，不要惊慌，冷静下来，一步一步地处理。

1.1 开始排障

在遭遇 Monitor 故障时，首先回答下列几个问题：

Mon 进程在运行吗？

我们首先要确保 Mon 进程是在正常运行的。很多人往往忽略了这一点。

是否可以连接 **Mon Server**？

有时候我们开启了防火墙，导致无法与 Monitor 的 IP 或端口进行通信。尝试使用 `ssh` 连接服务器，如果成功，再尝试用其他工具（如 `telnet` ， `nc` 等）连接 monitor 的端口。

ceph -s 命令是否能运行并收到集群回复？

如果答案是肯定的，那么你的集群已启动并运行着。你可以认为如果已经形成法定人数，monitors 就只会响应 `status` 请求。

如果 `ceph -s` 阻塞了，并没有收到集群的响应且输出了很多 `fault` 信息，很可能此时你的 monitors 全部都 down 掉了或只有部分在运行（但数量不足以形成法定人数）。

ceph -s 没完成是什么情况？

如果你到目前为止没有完成前述的几步，请返回去完成。然后你需要 `ssh` 登录到服务器上并使用 monitor 的管理套接字。

1.2 使用 Mon 的管理套接字

通过管理套接字，你可以用 Unix 套接字文件直接与指定守护进程交互。这个文件位于你 Mon 节点的 `run` 目录下，默认配置它位于 `/var/run/ceph/ceph-mon.ID.asok` ，但要是改过配置就不一定在那里了。如果你在那里没找到它，请检查 `ceph.conf` 里是否配置了其它路径，或者用下面的命令获取：

```
ceph-conf --name mon.ID --show-config-value admin_socket
```

请牢记，只有在 Mon 运行时管理套接字才可用。Mon 正常关闭时，管理套接字会被删除；如果 Mon 不运行了、但管理套接字还存在，就说明 Mon 不是正常关闭的。不管怎样，Mon 没在运行，你就不能使用管理套接字， `ceph` 命令会返回类似 `Error 111: Connection Refused` 的错误消息。

访问管理套接字很简单，就是让 `ceph` 工具使用 `asok` 文件。对于 Dumpling 之前的版本，命令是这样的：

```
ceph --admin-daemon /var/run/ceph/ceph-mon.<id>.asok <command>
```

对于 Dumpling 及后续版本，你可以用另一个（推荐的）命令：

```
ceph daemon mon.<id> <command>
```

`ceph` 工具的 `help` 命令会显示管理套接字支持的其它命令。请仔细了解一下 `config get` 、 `config show` 、 `mon_status` 和 `quorum_status` 命令，在排除 Mon 故障时它们会很有用。

1.3 理解 MON_STATUS

当集群形成法定人数后，或在没有形成法定人数时通过管理套接字， 用 `ceph` 工具可以获得 `mon_status` 信息。命令会输出关于 monitor 的大多数信息，包括部分 `quorum_status` 命令的输出内容。

下面是 `mon_status` 的输出样例：

```
{
    "name": "c",
      "rank": 2,
      "state": "peon",
      "election_epoch": 38,
      "quorum": [
        1,
        2
    ],
      "outside_quorum": [],
      "extra_probe_peers": [],
      "sync_provider": [],
      "monmap": {
        "epoch": 3,
          "fsid": "5c4e9d53-e2e1-478a-8061-f543f8be4cf8",
          "modified": "2013-10-30 04:12:01.945629",
          "created": "2013-10-29 14:14:41.914786",
          "mons": [
            {
                "rank": 0,
                  "name": "a",
                  "addr": "127.0.0.1:6789\/0"
            },
            {
                "rank": 1,
                  "name": "b",
                  "addr": "127.0.0.1:6790\/0"
            },
            {
                "rank": 2,
                  "name": "c",
                  "addr": "127.0.0.1:6795\/0"
            }
        ]
    }
}
```

从上面的信息可以看出，monmap 中包含 3 个monitor （*a*，*b* 和 *c*），只有 2 个 monitor 形成了法定人数，*c* 是法定人数中的 *peon* 角色（非 *leader* 角色）。

还可以看出， **a** 并不在法定人数之中。请看 `quorum` 集合。在集合中只有 *1* 和 *2* 。这不是 monitor 的名字，而是它们加入当前 monmap 后确定的等级。丢失了等级 0 的 monitor，根据 monmap ，这个 monitor 就是 `mon.a` 。

那么，monitor 的等级是如何确定的？

当加入或删除 monitor 时，会（重新）计算等级。计算时遵循一个简单的规则： `IP:PORT` 的组合值越大， 等级越低（等级数字越大，级别越低）。因此在上例中， `127.0.0.1:6789` 比其他 `IP:PORT` 的组合值都小，所以 `mon.a` 的等级是 0 。

1.4 最常见的 Mon 问题

达到了法定人数但是有至少一个 Monitor 处于 Down 状态

当此种情况发生时，根据你运行的 Ceph 版本，可能看到类似下面的输出：

```
root@OPS-ceph1:~# ceph health detail
HEALTH_WARN 1 mons down, quorum 1,2 b,c
mon.a (rank 0) addr 127.0.0.1:6789/0 is down (out of quorum)
```

如何解决？

首先，确认 mon.a 进程是否运行。

其次，确定可以从其他 monitor 节点连到 `mon.a` 所在节点。同时检查下端口。如果开了防火墙，还需要检查下所有 monitor 节点的 `iptables` ，以确定没有丢弃/拒绝连接。

如果前两步没有解决问题，请继续往下走。

首先，通过管理套接字检查问题 monitor 的 `mon_status` 。考虑到该 monitor 并不在法定人数中，它的状态应该是 `probing` ， `electing` 或 `synchronizing` 中的一种。如果它恰巧是 `leader` 或 `peon` 角色，它会认为自己在法定人数中，但集群中其他 monitor 并不这样认为。或者在我们处理故障的过程中它加入了法定人数，所以再次使用 `ceph -s` 确认下集群状态。如果该 monitor 还没加入法定人数，继续。

`probing` 状态是什么情况？

这意味着该 monitor 还在搜寻其他 monitors 。每次你启动一个 monitor，它会去搜寻 `monmap` 中的其他 monitors ，所以会有一段时间处于该状态。此段时间的长短不一。例如，单节点 monitor 环境， monitor 几乎会立即通过该阶段。在多 monitor 环境中，monitors 在找到足够的节点形成法定人数之前，都会处于该状态，这意味着如果 3 个 monitor 中的 2 个 down 了，剩下的 1 个会一直处于该状态，直到你再启动一个 monitor 。

如果你的环境已经形成法定人数，只要 monitor 之间可以互通，新 monitor 应该可以很快搜寻到其他 monitors 。如果卡在 probing 状态，并且排除了连接的问题，那很有可能是该 monitor 在尝试连接一个错误的 monitor 地址。可以根据 `mon_status` 命令输出中的 `monmap` 内容，检查其他 monitor 的地址是否和实际相符。如果不相符，请跳至恢复 **Monitor** 损坏的 **monmap**。如果相符，这些 monitor 节点间可能存在严重的时钟偏移问题，请首先参考时钟偏移，如果没有解决问题，可以搜集相关的日志并向社区求助。

`electing` 状态是什么情况？

这意味着该 monitor 处于选举过程中。选举应该很快就可以完成，但偶尔也会卡住，这往往是 monitors 节点时钟偏移的一个标志，跳转至时钟偏移获取更多信息。如果时钟是正确同步的，可以搜集相关日志并向社区求助。此种情况除非是一些（非常）古老的 bug ，往往都是由时钟不同步引起的。

`synchronizing` 状态是什么情况？

这意味着该 monitor 正在和集群中的其他 monitor 进行同步以便加入法定人数。Monitor 的数据库越小，同步过程的耗时就越短。

然而，如果你注意到 monitor 的状态从 synchronizing 变为 electing 后又变回 synchronizing ，那么就有问题了：集群的状态更新的太快（即产生新的 maps ），同步过程已经无法追赶上了。这种情况在早期版本中可以见到，但现在经过代码重构和增强，在较新版本中已经基本见不到了。

`leader` 或 **`peon`** 状态是什么情况？

这种情况不应该发生，但还是有一定概率会发生，这常和时钟偏移有关。如果你并没有时钟偏移的问题，请搜集相关的日志并向社区求助。

恢复 **Monitor** 损坏的 **monmap**

monmap 通常看起来是下面的样子，这取决于 monitor 的个数：

```
epoch 3
fsid 5c4e9d53-e2e1-478a-8061-f543f8be4cf8
last_changed 2013-10-30 04:12:01.945629
created 2013-10-29 14:14:41.914786
0: 127.0.0.1:6789/0 mon.a
1: 127.0.0.1:6790/0 mon.b
2: 127.0.0.1:6795/0 mon.c
```

不过也不一定就是这样的内容。比如，在早期版本的 Ceph 中，有一个严重 bug 会导致 `monmap` 的内容全为 0 。这意味着即使用 `monmaptool` 也不能读取它，因为全 0 的内容没有任何意义。另外一些情况，某个 monitor 所持有的 monmap 已严重过期，以至于无法搜寻到集群中的其他 monitors 。在这些状况下，你有两种可行的解决方法：

销毁 **monitor** 然后新建

只有在你确定不会丢失保存在该 monitor 上的数据时，你才能够采用这个方法。也就是说，集群中还有其他运行正常的 monitors，以便新 monitor 可以和其他 monitors 达到同步。请谨记，销毁一个 monitor 时，如果没有其上数据的备份，可能会丢失数据。

给 **monitor** 手动注入 **monmap**

通常是最安全的做法。你应该从剩余的 monitor 中抓取 monmap，然后手动注入 monmap 有问题的 monitor 节点。

下面是基本步骤：

1、是否已形成法定人数？如果是，从法定人数中抓取 monmap ：

```
ceph mon getmap -o /tmp/monmap
```

2、没有形成法定人数？直接从其他 monitor 节点上抓取 monmap （这里假定你抓取 monmap 的 monitor 的 id 是 ID-FOO 并且守护进程已经停止运行）：

```
ceph-mon -i ID-FOO --extract-monmap /tmp/monmap
```

3、停止你想要往其中注入 monmap 的 monitor。

4、注入 monmap 。

```
ceph-mon -i ID --inject-monmap /tmp/monmap
```

5、启动 monitor 。

请记住，能够注入 monmap 是一个很强大的特性，如果滥用可能会对 monitor 造成大破坏，因为这样做会覆盖 monitor 持有的最新 monmap 。

时钟偏移

Monitor 节点间明显的时钟偏移会对 monitor 造成严重的影响。这通常会导致一些奇怪的问题。为了避免这些问题，在 monitor 节点上应该运行时间同步工具。

允许的最大时钟偏移量是多少？

默认最大允许的时钟偏移量是 `0.05 秒` 。

如何增加最大时钟偏移量？

通过 mon-clock-drift-allowed 选项来配置。尽管你 可以 修改但不代表你 应该 修改。时钟偏移机制之所以是合理的，是因为有时钟偏移的 monitor 可能会表现不正常。未经测试而修改该值，尽管没有丢失数据的风险，但仍可能会对 monitors 的稳定性和集群的健康造成不可预知的影响。

如何知道是否存在时钟偏移？

Monitor 会用 `HEALTH_WARN` 的方式警告你。 `ceph health detail` 应该输出如下格式的信息：

```
mon.c addr 10.10.0.1:6789/0 clock skew 0.08235s > max 0.05s (latency 0.0045s)
```

这表示 `mon.c` 已被标记出正在遭受时钟偏移。

如果存在时钟偏移该怎么处理？

同步各 monitor 节点的时钟。运行 NTP 客户端会有帮助。如果你已经启动了 NTP 服务，但仍遭遇此问题，检查一下使用的 NTP 服务器是否离你的网络太过遥远，然后可以考虑在你的网络环境中运行自己的 NTP 服务器。最后这种选择可趋于减

少 monitor 时钟偏移带来的问题。

客户端无法连接或挂载

检查 IP 过滤表。某些操作系统安装工具会给 `iptables` 增加一条 `REJECT` 规则。这条规则会拒绝所有尝试连接该主机的客户端（除了 `ssh` ）。如果你的 monitor 主机设置了这条防火墙 `REJECT` 规则，客户端从其他节点连接过来时就会超时失败。你需要定位出拒绝客户端连接 Ceph 守护进程的那条 `iptables` 规则。比如，你需要对类似于下面的这条规则进行适当处理：

```
REJECT all -- anywhere anywhere reject-with icmp-host-prohibited
```

你还需要给 Ceph 主机的 IP 过滤表增加规则，以确保客户端可以访问 Ceph monitor （默认端口 6789 ）和 Ceph OSD （默认 6800 ~ 7300 ）的相关端口。

```
iptables -A INPUT -m multiport -p tcp -s {ip-address}/{netmask}
--dports 6789,6800:7300 -j ACCEPT
```

或者，如果你的环境允许，也可以直接关闭主机的防火墙。

1.5 Monitor 数据库失败

数据库崩溃的表现

Ceph monitor 把集群的各种 map 信息存放在 key/value 数据库中，如 LevelDB 。如果 monitor 是因为数据库崩溃而失败，在 monitor 的 log 日志中应该会有如下错误信息：

```
Corruption: error in middle of record
```

或：

```
Corruption: 1 missing files; e.g.: /var/lib/ceph/mon/mon.0/store
.db/1234567.ldb
```

通过健康的 Monitor(s) 恢复

如果还有幸存的 monitor，我们通常可以用新的数据库替换崩溃的数据库。并且在启动后，新加入的成员会和其他健康的伙伴进行同步，一旦同步完成，它就可以为客户端提供服务了。

通过 OSDs 恢复

但是万一所有的 monitors 都同时失败了该怎么办？由于建议用户在部署集群时至少安装 3 个 monitors，同时失效的可能性较小。但是数据中心意外的断电，再加上磁盘/文件系统配置不当，可能会引起底层文件系统失败，从而杀掉所有的 monitors 。这种情况下，我们可以通过存放在 OSDs 上的信息来恢复 monitor 的数据库：

```
ms=/tmp/mon-store
mkdir $ms
# collect the cluster map from OSDs
for host in $hosts; do
      rsync -avz $ms user@host:$ms
      rm -rf $ms
      ssh user@host <<EOF
        for osd in /var/lib/osd/osd-*; do
              ceph-objectstore-tool --data-path $osd --op update
-mon-db --mon-store-path $ms
        done
      EOF
      rsync -avz user@host:$ms $ms
done
# rebuild the monitor store from the collected map, if the clust
er does not
# use cephx authentication, we can skip the following steps to u
pdate the
# keyring with the caps, and there is no need to pass the "--key
ring" option.
# i.e. just use "ceph-monstore-tool /tmp/mon-store rebuild" inst
ead
ceph-authtool /path/to/admin.keyring -n mon. \
  --cap mon allow 'allow *'
ceph-authtool /path/to/admin.keyring -n client.admin \
  --cap mon allow 'allow *' --cap osd 'allow *' --cap mds 'allow
 *'
ceph-monstore-tool /tmp/mon-store rebuild -- --keyring /path/to/
admin.keyring
# backup corrupted store.db just in case
mv /var/lib/ceph/mon/mon.0/store.db /var/lib/ceph/mon/mon.0/stor
e.db.corrupted
mv /tmp/mon-store/store.db /var/lib/ceph/mon/mon.0/store.db
```

上面的这些步骤：

1. 从所有的 OSD 主机上收集 map 信息。
2. 重建数据库。
3. 用恢复副本替换 mon.0 上崩溃的数据库。

已知的限制

下面这些信息无法通过上述步骤恢复：

- 一些新增的 **keyring** ：通过 `ceph auth add` 命令增加的所有 OSD keyrings 都可以恢复。用 `ceph-monstore-tool` 可以导入 `client.admin` 的 keyring 。但是 MDS 和其他 keyrings 在被恢复的那个 monitor 数据库中就会丢失。你可能需要手动重新添加一下。
- **pg** 的设置：通过 `ceph pg set_full_ratio` 和 `ceph pg set_nearfull_ratio` 命令设置的 `full ratio` 和 `nearfull ratio` 值会丢失。
- **MDS Maps**：MDS maps 会丢失。

磁盘空间不足导致 MON DOWN

当 monitor 进程检测到本地可用磁盘空间不足时，会停止 monitor 服务。Monitor 的日志中应该会有类似如下信息的输出：

```
2016-09-01 16:45:54.994488 7fb1cac09700  0 mon.jyceph01@0(leader
).data_health(62) update_stats avail 5% total 297 GB, used 264 G
B, avail 18107 MB
2016-09-01 16:45:54.994747 7fb1cac09700 -1 mon.jyceph01@0(leader
).data_health(62) reached critical levels of available space on
local monitor storage -- shutdown!
```

清理本地磁盘，增大可用空间，重启 monitor 进程，即可恢复正常。

通过 Mon 数据库的备份恢复

具体请参考本手册第三部分 2. Monitor 的备份和恢复 。

2. 常见 OSD 故障处理

进行 OSD 排障前，先检查一下 monitors 和网络。如果 `ceph health` 或 `ceph -s` 返回的是健康状态，这意味着 monitors 形成了法定人数。如果 monitor 还没达到法定人数、或者 monitor 状态错误，要先解决 monitor 的问题。核实下你的网络，确保它在正常运行，因为网络对 OSD 的运行和性能有显著影响。

2.1 收集 OSD 数据

开始 OSD 排障的第一步最好先收集信息，另外还有监控 OSD 时收集的，如 `ceph osd tree` 。

Ceph 日志

如果你没改默认路径，可以在 `/var/log/ceph` 下找到 Ceph 的日志：

```
ls /var/log/ceph
```

如果看到的日志还不够详细，可以增大日志级别。请参考1.12 日志和调试，查阅如何保证看到大量日志又不影响集群运行。

管理套接字

用管理套接字工具检索运行时信息。列出节点上所有 Ceph 套接字：

```
ls /var/run/ceph
```

然后，执行下例命令显示可用选项，把 `{daemon-name}` 换成实际的守护进程（如 osd.0 ）：

```
ceph daemon osd.0 help
```

或者，你也可以指定一个 `{socket-file}` （如 `/var/run/ceph` 下的文件）：

```
ceph daemon {socket-file} help
```

和其它手段相比，管理套接字允许你：

- 在运行时列出配置
- 列出历史操作
- 列出操作的优先队列状态
- 列出在进行的操作
- 列出性能计数器

显示可用空间

可能会引起文件系统问题。用 `df` 命令显示文件系统的可用空间。

```
df -h
```

其它用法见 `df --help` 。

I/O 统计信息

用 `iostat` 工具定位 I/O 相关问题。

```
iostat -x
```

诊断信息

要查看诊断信息，配合 `less` 、 `more` 、 `grep` 或 `tail` 使用 `dmesg` ，例如：

```
dmesg | grep scsi
```

2.2 停止数据向外重平衡

你得周期性地对集群的子集进行维护，或解决某个故障域的问题（如某个机架）。如果你不想在停机维护 OSD 时让 CRUSH 自动重均衡，首先设置集群的 `noout` 标志：

```
ceph osd set noout
```

设置了 noout 后，你就可以停机维护失败域内的 OSD 了。

```
stop ceph-osd id={num}
```

注意：在定位某故障域内的问题时，停机的 OSD 内的 PG 状态会变为 `degraded` 。

维护结束后，重启 OSD 。

```
start ceph-osd id={num}
```

最后，解除 `noout` 标志。

```
ceph osd unset noout
```

2.3 OSD 没运行

通常情况下，简单地重启 `ceph-osd` 进程就可以让它重回集群并恢复。

OSD 起不来

如果你重启了集群，但其中一个 OSD 起不来，依次检查：

- 配置文件： 如果你新装的 OSD 不能启动，检查下配置文件，确保它符合规定（比如 `host` 而非 `hostname` ，等等）。
- 检查路径： 检查配置文件的路径，以及 OSD 的数据和日志分区路径。如果你分离了 OSD 的数据和日志分区、而配置文件和实际挂载点存在出入，启动 OSD 时就可能失败。如果你想把日志存储于一个块设备，应该为日志硬盘分区并为各 OSD 分别指定一个分区。

- 检查最大线程数：如果你的节点有很多 OSD ，也许就会触碰到默认的最大线程数限制（如通常是 32k 个），尤其是在恢复期间。你可以用 `sysctl` 增大线程数，把最大线程数更改为支持的最大值（即 4194303 ），看看是否有用。例如：

  sysctl -w kernel.pid_max=4194303

如果增大最大线程数解决了这个问题，你可以把此配置 `kernel.pid_max` 写入配置文件 `/etc/sysctl.conf`，使之永久生效，例如：

```
kernel.pid_max = 4194303
```

- 内核版本：确认你使用的内核版本和发布版本。Ceph 默认依赖一些第三方工具，这些工具可能有缺陷或者与特定发布版和/或内核版本冲突（如 Google perftools ）。检查下操作系统推荐表，确保你已经解决了内核相关的问题。
- 段错误：如果有了段错误，提高日志级别（如果还没提高），再试试。如果重现了，联系 ceph-devel 邮件列表并提供你的配置文件、monitor 输出和日志文件内容。

OSD 失败

当 `ceph-osd` 挂掉时，monitor 可通过活着的 `ceph-osd` 了解到此情况，并通过 `ceph health` 命令报告：

```
ceph health
HEALTH_WARN 1/3 in osds are down
```

特别地，有 `ceph-osd` 进程标记为 `in` 且 `down` 的时候，你也会得到警告。你可以用下面的命令得知哪个 `ceph-osd` 进程挂了：

```
ceph health detail
HEALTH_WARN 1/3 in osds are down
osd.0 is down since epoch 23, last address 192.168.106.220:6800/
11080
```

如果有硬盘失败或其它错误使 `ceph-osd` 不能正常运行或重启，将会在日志文件 `/var/log/ceph/` 里输出一条错误信息。

如果守护进程因心跳失败、或者底层核心文件系统无响应而停止，查看 `dmesg` 获取硬盘或者内核错误。

如果是软件错误（失败的断言或其它意外错误），应该向 ceph-devel 邮件列表报告。

硬盘没剩余空间

Ceph 不允许你向满的 OSD 写入数据，以免丢失数据。在运行着的集群中，你应该能收到集群空间将满的警告。 `mon osd full ratio` 默认为 0.95 ，或达到 95% 的空间使用率时它将阻止客户端写入数据。 `mon osd nearfull ratio` 默认为 0.85 ，也就是说达到容量的 85% 时它会产生健康警告。

满载集群问题一般产生于测试小型 Ceph 集群上如何处理 OSD 失败时。当某一节点使用率较高时，集群能够很快掩盖将满和占满率。如果你在测试小型集群上的 Ceph 如何应对 OSD 失败，应该保留足够的可用磁盘空间，然后试着临时降低 `mon osd full ratio` 和 `mon osd nearfull ratio` 值。

`ceph health` 会报告将满的 `ceph-osds` ：

```
ceph health
HEALTH_WARN 1 nearfull osds
osd.2 is near full at 85%
```

或者：

```
ceph health
HEALTH_ERR 1 nearfull osds, 1 full osds
osd.2 is near full at 85%
osd.3 is full at 97%
```

处理这种情况的最好方法就是在出现 `near full` 告警时尽快增加新的 `ceph-osd` ，这允许集群把数据重分布到新 OSD 里。

如果因满载而导致 OSD 不能启动，你可以试着删除那个 OSD 上的一些数据。但是这时有个问题，当一个 OSD 使用比例达到 95% 时，集群将不接受任何 Ceph Client 端的读写数据的请求。这时 `rbd rm` 删除命令将不会得到响应。

让集群能够读写是首先要做的事情。最容易想到的就是调高 `mon osd full ratio` 和 `mon osd nearfull ratio` 值，但是对于生产环境，一旦调整这个全局比例，可能会导致整个集群的数据都会动起来，引发更多的数据迁移。因此另一种折衷方法就是单独调整已满 OSD 的 near full 和 full 比例；也可以使用调低 OSD 的 crush weight 的方法，使已满 OSD 上的数据迁移一部分出去。

```
# 调整单个 osd 的比例
ceph tell osd.id injectargs '--mon-osd-full-ratio .98'
ceph tell osd.id injectargs '--mon-osd-full-ratio 0.98'

# 调整 osd 的 crush weight 值
ceph osd crush reweight osd.id {a-little-lower-weight-value}
```

2.4 OSD 龟速或无响应

一个反复出现的问题是 OSD 龟速或无响应。在深入性能问题前，你应该先确保不是其他故障。例如，确保你的网络运行正常、且 OSD 在运行，还要检查 OSD 是否被恢复流量拖住了。

Tip： 较新版本的 Ceph 能更好地处理恢复，可防止恢复进程耗尽系统资源而导致 `up` 且 `in` 的 OSD 不可用或响应慢。

网络问题

Ceph 是一个分布式存储系统，所以它依赖于网络来互联 OSD 们、复制对象、从错误中恢复和检查心跳。网络问题会导致 OSD 延时和震荡（反复经历 up and down，详情可参考下文中的相关小节） 。

确保 Ceph 进程和 Ceph 依赖的进程已建立连接和/或在监听。

```
netstat -a | grep ceph
netstat -l | grep ceph
sudo netstat -p | grep ceph
```

检查网络统计信息。

```
netstat -s
```

驱动器配置

一个存储驱动器应该只用于一个 OSD 。如果有其它进程共享驱动器，顺序读写吞吐量会成为瓶颈，包括日志、操作系统、monitor 、其它 OSD 和非 Ceph 进程。

Ceph 在日志记录完成之后才会确认写操作，所以使用 `ext4` 或 `XFS` 文件系统时高速的 SSD 对降低响应延时很有吸引力。与之相比， `btrfs` 文件系统可以同时读写日志和数据分区。

注意： 给驱动器分区并不能改变总吞吐量或顺序读写限制。把日志分离到单独的分区可能有帮助，但最好是另外一块硬盘的分区。

扇区损坏 / 碎片化硬盘

检修下硬盘是否有坏扇区和碎片。这会导致总吞吐量急剧下降。

MON 和 OSD 共存

Monitor 通常是轻量级进程，但它们会频繁调用 `fsync()` ，这会妨碍其它工作负载，特别是 Mon 和 OSD 共享驱动器时。另外，如果你在 OSD 主机上同时运行 Mon，遭遇的性能问题可能和这些相关：

- 运行较老的内核（低于 3.0 ）
- Argonaut 版运行在老的 glibc 之上
- 运行的内核不支持 syncfs(2) 系统调用

在这些情况下，同一主机上的多个 OSD 会相互拖垮对方。它们经常导致爆炸式写入。

进程共存

共用同一套硬件、并向 Ceph 写入数据的进程（像基于云的解决方案、虚拟机和其他应用程序）会导致 OSD 延时大增。一般来说，我们建议用单独的主机跑 Ceph 、其它主机跑其它进程。实践证明把 Ceph 和其他应用程序分开可提高性能、并简化故障排除和维护。

日志记录级别

如果你为追踪某问题提高过日志级别，结束后又忘了调回去，这个 OSD 将向硬盘写入大量日志。如果你想始终保持高日志级别，可以考虑给默认日志路径（即 `/var/log/ceph/$cluster-$name.log` ）挂载一个单独的硬盘。

恢复限流

根据你的配置， Ceph 可以降低恢复速度来维持性能，否则它会加快恢复速度而影响 OSD 的性能。检查下 OSD 是否正在恢复。

内核版本

检查下你在用的内核版本。较老的内核也许没有反合能提高 Ceph 性能的代码。

内核与 SYNCFS 问题

试试在一个主机上只运行一个 OSD ，看看能否提升性能。老内核未必支持有 `syncfs(2)` 系统调用的 `glibc` 。

文件系统问题

当前，我们推荐基于 `xfs` 部署集群。 `btrfs` 有很多诱人的功能，但文件系统内的缺陷可能会导致性能问题。我们不推荐使用 `ext4` ，因为 xattr 大小的限制破坏了对长对象名的支持（ RGW 需要）。

内存不足

我们建议为每 OSD 进程规划 1GB 内存。你也许注意到了，通常情况下 OSD 仅会使用一小部分（如 100 - 200MB ）。你也许想用这些空闲内存跑一些其他应用，如虚拟机等等。然而当 OSD 进入恢复状态时，其内存利用率将激增。如果没有足够的可用内存，此 OSD 的性能将会明显下降。

OLD REQUESTS 或 SLOW REQUESTS

如果某 `ceph-osd` 守护进程对一请求响应很慢，它会生成日志消息来抱怨请求耗费的时间过长。默认警告阈值是 30 秒，可以通过 `osd op complaint time` 选项来配置。这种情况发生时，集群日志会收到这些消息。

很老的版本抱怨 “old requests” ：

```
osd.0 192.168.106.220:6800/18813 312 : [WRN] old request osd_op(
client.5099.0:790 fatty_26485_object789 [write 0~4096] 2.5e54f64
3) v4 received at 2012-03-06 15:42:56.054801 currently waiting f
or sub ops
```

较新版本的 Ceph 抱怨 “slow requests” ：

```
{date} {osd.num} [WRN] 1 slow requests, 1 included below; oldest
 blocked for > 30.005692 secs
{date} {osd.num}  [WRN] slow request 30.005692 seconds old, rece
ived at {date-time}: osd_op(client.4240.0:8 benchmark_data_ceph-
1_39426_object7 [write 0~4194304] 0.69848840) v4 currently waiti
ng for subops from [610]
```

可能的原因有：

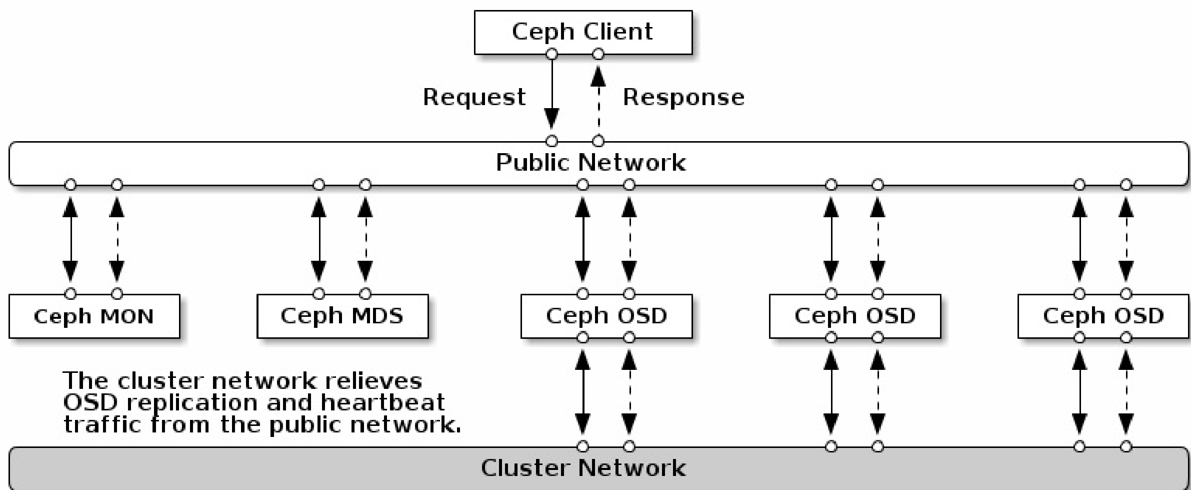
- 坏驱动器（查看 `dmesg` 输出）
- 内核文件系统缺陷（查看 `dmesg` 输出）
- 集群过载（检查系统负载、 iostat 等等）
- ceph-osd 守护进程的 bug

可能的解决方法：

- 从 Ceph 主机分离 VM 云解决方案
- 升级内核
- 升级 Ceph
- 重启 OSD

2.5 震荡的 OSD

我们建议同时部署 public（前端）网络和 cluster（后端）网络，这样能更好地满足对象复制的网络性能需求。另一个优点是你可以运营一个不连接互联网的集群，以此避免某些拒绝服务攻击。 OSD 们互联和检查心跳时会优选 cluster（后端）网络。



然而，如果 cluster（后端）网络失败、或出现了明显的延时，同时 public（前端）网络却运行良好，OSD 目前不能很好地处理这种情况。这时 OSD 们会向 monitor 报告邻居 `down` 了、同时报告自己是 `up` 的，我们把这种情形称为震荡（flapping）。

如果有原因导致 OSD 震荡（反复地被标记为 `down` ，然后又 `up` ），你可以强制 monitor 停止这种震荡状态：

```
ceph osd set noup      # prevent OSDs from getting marked up
ceph osd set nodown    # prevent OSDs from getting marked down
```

这些标记记录在 osdmap 数据结构里：

```
ceph osd dump | grep flags
flags no-up,no-down
```

可用下列命令清除标记：

```
ceph osd unset noup
ceph osd unset nodown
```

Ceph 还支持另外两个标记 `noin` 和 `noout` ，它们可防止正在启动的 OSD 被标记为 `in` （可以分配数据），或被误标记为 `out` （不管 `mon osd down out interval` 的值是多少）。

注意： `noup` 、 `noout` 和 `nodown` 从某种意义上说是临时的，一旦标记被清除了，被它们阻塞的动作短时间内就会发生。另一方面， `noin` 标记阻止 OSD 启动后加入集群，但其它守护进程都维持原样。

3. 常见 PG 故障处理

3.1 PG 无法达到 CLEAN 状态

创建一个新集群后，PG 的状态一直处于 `active` ， `active + remapped` 或 `active + degraded` 状态， 而无法达到 `active + clean` 状态 ，那很可能是你的配置有问题。

你可能需要检查下集群中有关 Pool 、 PG 和 CRUSH 的配置项，做以适当的调整。

一般来说，你的集群中需要多于 1 个 OSD，并且存储池的 size 要大于 1 副本。

单节点集群

有时候，我们需要搭建一个单节点的 Ceph 实验环境。此时，在开始创建 monitor 和 OSD 之前，你需要把 Ceph 配置文件中的 `osd crush chooseleaf type` 选项从默认值 `1` （表示 `host` 或 `node` ）修改为 `0` （表示 `osd` ）。这样做是告诉 Ceph 允许把数据的不同副本分布到同一 host 的 OSDs 上。

OSD 个数小于副本数

如果你已经启动了 2 个 OSD，它们都处于 `up` 和 `in` 的状态，但 PG 仍未达到 `active + clean` 状态，那可能是给 `osd pool default size` 设置了一个大于 `2` 的值。

如果你想要在 `active + degraded` 状态（ 2 副本）操作你的集群，可以设置 `osd pool default min size` 为 2 ，这样你就可以对处于 `active + degraded` 的对象写入数据。然后你还可以把 `osd pool default size` 的值改为 2 ，这样集群就可以达到 `active + clean` 状态了。

另外，修改参数 `osd pool default size/min_size` 后，只会对后面新建的 pool 起作用。如果想修改已存在的 pool 的 `size/min_size` ，可用下面的命令：

```
ceph osd pool set <poolname> size|min_size <val>
```

注意：你可以在运行时修改参数值。如果是在 Ceph 配置文件中进行的修改，你可能需要重启集群。

POOL SIZE = 1

如果你设置了 `osd pool default size` 的值为 `1` ，那你就仅有对象的单份拷贝。OSD 依赖于其他 OSD 告诉自己应该保存哪些对象。如果第一个 OSD 持有对象的拷贝，并且没有第二份拷贝，那么也就没有第二个 OSD 去告诉第一个 OSD 它应该保管那份拷贝。对于每一个映射到第一个 OSD 上的 PG （参考 `ceph pg dump` 的输出），你可以强制第一个 OSD 关注它应该保存的 PGs ：

```
ceph pg force_create_pg <pgid>
```

CRUSH MAP 错误

PG 达不到 clean 状态的另一个可能的原因就是集群的 CRUSH Map 有错误，导致 PG 不能映射到正确的地方。

3.2 卡住的 PGs

有失败发生后，PG 会进入“degraded”（降级）或“peering”（连接建立中）状态，这种情况时有发生。通常这些状态意味着正常的失败恢复正在进行。然而，如果一个 PG 长时间处于这些状态中的某个，就意味着有更大的问题。因此 monitor 在 PG 卡 （ stuck ） 在非最优状态时会告警。我们具体检查：

- `inactive` （不活跃）—— PG 长时间不是 `active` （即它不能提供读写服务了）；
- `unclean` （不干净）—— PG 长时间不是 `clean` （例如它未能从前面的失败完全恢复）；
- `stale` （不新鲜）—— PG 状态没有被 `ceph-osd` 更新，表明存储这个 PG 的所有节点可能都 `down` 了。

你可以用下列命令显式地列出卡住的 PGs：

```
ceph pg dump_stuck stale
ceph pg dump_stuck inactive
ceph pg dump_stuck unclean
```

卡在 `stale` 状态的 PG 通过重启 ceph-osd 进程通常可以修复；卡在 `inactive` 状态的 PG 通常是互联问题（参见 ***PG* 挂了 —— 互联失败** ）；卡在 `unclean` 状态的 PG 通常是由于某些原因阻止了恢复的完成，像未找到的对象（参见 未找到的对象 ）。

3.3 PG 挂了 —— 互联失败

在某些情况下， `ceph-osd` 互联进程会遇到问题，阻值 PG 达到活跃、可用的状态。例如， `ceph health` 也许显示：

```
ceph health detail
HEALTH_ERR 7 pgs degraded; 12 pgs down; 12 pgs peering; 1 pgs re
covering; 6 pgs stuck unclean; 114/3300 degraded (3.455%); 1/3 i
n osds are down
...
pg 0.5 is down+peering
pg 1.4 is down+peering
...
osd.1 is down since epoch 69, last address 192.168.106.220:6801/
8651
```

可以查询到 PG 为何被标记为 `down` ：

```
ceph pg 0.5 query

{ "state": "down+peering",
    ...
    "recovery_state": [
       { "name": "Started\/Primary\/Peering\/GetInfo",
          "enter_time": "2012-03-06 14:40:16.169679",
          "requested_info_from": []},
          { "name": "Started\/Primary\/Peering",
          "enter_time": "2012-03-06 14:40:16.169659",
          "probing_osds": [
                  0,
               1],
          "blocked": "peering is blocked due to down osds",
          "down_osds_we_would_probe": [
               1],
          "peering_blocked_by": [
                  { "osd": 1,
                  "current_lost_at": 0,
                  "comment": "starting or marking this osd lost
may let us proceed"}]},
          { "name": "Started",
          "enter_time": "2012-03-06 14:40:16.169513"}
  ]
}
```

`recovery_state` 段告诉我们互联过程因 `ceph-osd` 进程挂了而被阻塞，本例是 `osd.1` 挂了，启动这个进程应该就可以恢复。

或者，如果 `osd.1` 发生了灾难性的失败（如硬盘损坏），我们可以告诉集群它丢失（ `lost` ）了，让集群尽力完成副本拷贝。

重要： 集群不能保证其它数据副本是一致且最新的，就会很危险！

让 Ceph 无论如何都继续：

```
ceph osd lost 1
```

恢复将继续进行。

3.4 未找到的对象

某几种失败相组合，可能导致 Ceph 抱怨有找不到（ `unfound` ）的对象：

```
ceph health detail
HEALTH_WARN 1 pgs degraded; 78/3778 unfound (2.065%)
pg 2.4 is active+degraded, 78 unfound
```

这意味着存储集群知道一些对象（或者存在对象的较新副本）存在，却没有找到它们的副本。下例展示了这种情况是如何发生的，一个 PG 的数据存储在 ceph-osd 1 和 2 上：

- 1 挂了
- 2 独自处理一些写动作
- 1 起来了
- 1 和 2 重新互联，1 上面丢失的对象加入队列准备恢复
- 新对象还未拷贝完，2 挂了

这时，1 知道这些对象存在，但是活着的 `ceph-osd` 都没有这些副本。这种情况下，读写这些对象的 IO 就会被阻塞，集群只能指望 down 掉的节点尽早恢复。这样处理是假设比直接给用户返回一个 IO 错误要好一些。

首先，你应该确认哪些对象找不到了：

```
ceph pg 2.4 list_missing [starting offset, in json]

{ "offset": { "oid": "",
     "key": "",
     "snapid": 0,
     "hash": 0,
     "max": 0},
"num_missing": 0,
"num_unfound": 0,
"objects": [
   { "oid": "object 1",
       "key": "",
     "hash": 0,
     "max": 0 },
   ...
],
"more": 0}
```

如果在一次查询里列出的对象太多， `more` 这个字段将为 `true` ，你就可以查询更多。

其次，你可以找出哪些 OSD 上探测到、或可能包含数据：

```
ceph pg 2.4 query

"recovery_state": [
     { "name": "Started\/Primary\/Active",
         "enter_time": "2012-03-06 15:15:46.713212",
         "might_have_unfound": [
             { "osd": 1,
                 "status": "osd is down"}]},
```

本例中，集群知道 `osd.1` 可能有数据，但它挂了（ `down` ）。所有可能的状态有：

- 已经探测到了
- 在查询
- OSD 挂了

- 尚未查询

有时候集群要花一些时间来查询可能的位置。

还有一种可能性，对象存在于其它位置却未被列出。例如，集群里的一个 ceph-osd 停止且被剔出集群，然后集群完全恢复了；后来一系列的失败导致了未找到的对象，它也不会觉得早已死亡的 ceph-osd 上仍可能包含这些对象。（这种情况几乎不太可能发生）。

如果所有可能的位置都查询过了但仍有对象丢失，那就得放弃丢失的对象了。这仍可能是罕见的失败组合导致的，集群在写操作恢复后，未能得知写入是否已执行。以下命令把未找到的（ `unfound` ）对象标记为丢失（ `lost` ）。

```
ceph pg 2.5 mark_unfound_lost revert|delete
```

上述最后一个参数告诉集群应如何处理丢失的对象。

- `delete` 选项将导致完全删除它们。
- `revert` 选项（纠删码存储池不可用）会回滚到前一个版本或者（如果它是新对象的话）删除它。要慎用，它可能迷惑那些期望对象存在的应用程序。

3.5 无家可归的 PG

拥有 PG 拷贝的 OSD 可能会全部失败，这种情况下，那一部分的对象存储不可用， monitor 也就不会收到那些 PG 的状态更新了。为检测这种情况，monitor 会把任何主 OSD 失败的 PG 标记为 `stale` （不新鲜），例如：

```
ceph health
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
```

可以找出哪些 PG 是 `stale` 状态，和存储这些归置组的最新 OSD ，命令如下：

```
ceph health detail
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
...
pg 2.5 is stuck stale+active+remapped, last acting [2,0]
...
osd.10 is down since epoch 23, last address 192.168.106.220:6800
/11080
osd.11 is down since epoch 13, last address 192.168.106.220:6803
/11539
osd.12 is down since epoch 24, last address 192.168.106.220:6806
/11861
```

如果想使 PG 2.5 重新上线，例如，上面的输出告诉我们它最后由 `osd.0` 和 `osd.2` 管理，重启这些 `ceph-osd` 将恢复之（可以假定还有其它的很多 PG 也会进行恢复）。

3.6 只有几个 OSD 接收数据

如果你的集群有很多节点，但只有其中几个接收数据，检查下存储池里的 PG 数量。因为 PG 是映射到多个 OSD 的，较少的 PG 将不能均衡地分布于整个集群。试着创建个新存储池，设置 PG 数量是 OSD 数量的若干倍。更详细的信息可以参考 Ceph 官方文档 —— Placement Groups 。

3.7 不能写入数据

如果你的集群已启动，但一些 OSD 没起来，导致不能写入数据，确认下运行的 OSD 数量满足 PG 要求的最低 OSD 数。如果不能满足， Ceph 就不会允许你写入数据，因为 Ceph 不能保证复制能如愿进行。这个最低 OSD 个数是由参数 `osd pool default min size` 限定的。

3.8 PG 不一致

如果收到 `active + clean + inconsistent` 这样的状态，很可能是由于在对 PG 做擦洗（ scrubbing ）时发生了错误。如果是由于磁盘错误导致的不一致，请检查磁盘，如果磁盘有损坏，可能需要将这个磁盘对应的 OSD 踢出集群，然后进行更换。生产环境中遇到过不一致的问题，就是由于磁盘坏道导致的。

当集群中出现 PG 不一致的问题时，执行 `ceph -s` 命令会出现下面的信息：

```
root@mon:~# ceph -s
    cluster 614e77b4-c997-490a-a3f9-e89aa0274da3
     health HEALTH_ERR
            1 pgs inconsistent
            1 scrub errors
     monmap e5: 1 mons at {osd1=10.95.2.43:6789/0}
            election epoch 796, quorum 0 osd1
     osdmap e1079: 3 osds: 3 up, 3 in
            flags sortbitwise
      pgmap v312153: 384 pgs, 6 pools, 1148 MB data, 311 objects
            3604 MB used, 73154 MB / 76759 MB avail
                 383 active+clean
                   1 active+clean+inconsistent
```

1、查找处于 `inconsistent` 状态的问题 PG ：

```
root@mon:~# ceph health detail
HEALTH_ERR 1 pgs inconsistent; 1 scrub errors
pg 9.14 is active+clean+inconsistent, acting [1,2,0]
1 scrub errors
```

这个有问题的 PG 分布在 `osd.1` 、 `osd.2` 和 `osd.0` 上，其中 `osd.1` 是主 OSD。

2、去主 OSD（ osd.1 ）的日志中查找不一致的具体对象 。

```
root@osd0:~# grep -Hn 'ERR' /var/log/ceph/ceph-osd.1.log
/var/log/ceph/ceph-osd.1.log:30:2016-11-10 13:49:07.848804 7f628
c5e6700 -1 log_channel(cluster) log [ERR] : 9.14 shard 0: soid 9
:29b4ad99:::rbd_data.1349f035c101d9.0000000000000001:head missin
g attr _
/var/log/ceph/ceph-osd.1.log:31:2016-11-10 13:49:07.849803 7f628
c5e6700 -1 log_channel(cluster) log [ERR] : 9.14 scrub 0 missing
, 1 inconsistent objects
/var/log/ceph/ceph-osd.1.log:32:2016-11-10 13:49:07.849824 7f628
c5e6700 -1 log_channel(cluster) log [ERR] : 9.14 scrub 1 errors
```

从日志中可以知道，是 `rbd_data.1349f035c101d9.0000000000000001` 这个对象的属性 `_` 丢失了，所以在 scrub 的过程中产生了 error 。

3、执行 `ceph pg repair` 命令修复问题 PG 。

```
root@mon:~# ceph pg repair 9.14
instructing pg 9.14 on osd.1 to repair
```

4、检查 Ceph 集群是否恢复到 `HEALTH_OK` 状态。

```
root@mon:~# ceph -s
    cluster 614e77b4-c997-490a-a3f9-e89aa0274da3
     health HEALTH_OK
     monmap e5: 1 mons at {osd1=10.95.2.43:6789/0}
            election epoch 796, quorum 0 osd1
     osdmap e1079: 3 osds: 3 up, 3 in
            flags sortbitwise
      pgmap v312171: 384 pgs, 6 pools, 1148 MB data, 311 objects
            3604 MB used, 73154 MB / 76759 MB avail
                 384 active+clean
```

osd.1 的日志里也提示修复成功：

```
2016-11-10 14:04:31.732640 7f628c5e6700  0 log_channel(cluster)
log [INF] : 9.14 repair starts
2016-11-10 14:04:31.827951 7f628edeb700 -1 log_channel(cluster)
log [ERR] : 9.14 shard 0: soid 9:29b4ad99:::rbd_data.1349f035c10
1d9.0000000000000001:head missing attr _
2016-11-10 14:04:31.828117 7f628edeb700 -1 log_channel(cluster)
log [ERR] : 9.14 repair 0 missing, 1 inconsistent objects
2016-11-10 14:04:31.828273 7f628edeb700 -1 log_channel(cluster)
log [ERR] : 9.14 repair 1 errors, 1 fixed
```

如果经过前面的步骤，Ceph 仍没有达到 `HEALTH_OK` 状态，可以尝试用下面这种方式进行修复。

1、停掉不一致的 object 所属的 osd 。

```
stop ceph-osd id=xxx
```

2、刷新该 osd 的日志。

```
ceph-osd -i xx --flush-journal
```

3、将不一致的 object 移除。

```
mv /var/lib/ceph/osd/ceph-{osd-id}/current/{pg.id}_head/ rbd\\ud
ata.xxx /home
```

4、重新启动该 osd 。

```
start ceph-osd id=xx
```

5、重新执行修复命令。

```
ceph pg repair {pg_id}
```

6、检查 Ceph 集群是否恢复到 `HEALTH_OK` 状态。

3.9 Too Many/Few PGs per OSD

有时候，我们在 ceph -s 的输出中可以看到如下的告警信息：

```
root@node241:~# ceph -s
    cluster 3b37db44-f401-4409-b3bb-75585d21adfe
     health HEALTH_WARN
            too many PGs per OSD (652 > max 300)
     monmap e1: 1 mons at {node241=192.168.2.41:6789/0}
            election epoch 1, quorum 0 node241
     osdmap e408: 5 osds: 5 up, 5 in
      pgmap v23049: 1088 pgs, 16 pools, 256 MB data, 2889 object
s
            6100 MB used, 473 GB / 479 GB avail
                 1088 active+clean
```

这是因为集群 OSD 数量较少，测试过程中建立了多个存储池，每个存储池都要建立一些 PGs 。而目前 Ceph 配置的默认值是每 OSD 上最多有 300 个 PGs 。在测试环境中，为了快速解决这个问题，可以调大集群的关于此选项的告警阀值。方法如下：

在 monitor 节点的 ceph.conf 配置文件中添加:

```
[global]
.......
mon_pg_warn_max_per_osd = 1000
```

然后重启 monitor 进程。

或者直接用 `tell` 命令在运行时更改参数的值而不用重启服务：

```
ceph tell mon.* injectargs '--mon_pg_warn_max_per_osd 1000'
```

而另一种情况， `too few PGs per OSD （16 < min 20）` 这样的告警信息则往往出现在集群刚刚建立起来，除了默认的 rbd 存储池，还没建立自己的存储池，再加上 OSD 个数较多，就会出现这个提示信息。这通常不是什么问题，也无需修改配置项，在建立了自己的存储池后，这个告警信息就会消失。

4. 全局Ceph节点宕机处理

在极端情况下，如数据中心断电，造成 Ceph 存储集群全局宕机，可以按照本节所示流程进行 Ceph 集群上电恢复操作。

4.1 手动上电执行步骤

1. 如为 Ceph 集群上电，monitor server 应最先上电；集群上电前确认使用 Ceph 之前端作业服务已停止。

2. 使用 IPMI 或于设备前手动进行上电。

3. 确认 NTP 服务及系统时间已同步，命令如下：

   `# ps-ef | grep ntp`

   `# date`

   `# ntpq -p`

4. 登入上电之 ceph server 确认 ceph service 已正常运行，命令如下：

   `# ps -ef | grep ceph`

5. 登入集群 monitor server 查看状态，OSD 全都 up 集群仍为 `noout flag(s) set`

   `# ceph -s`

   `# ceph osd tree`

6. 登入 monitor server 解除 `stopping w/out rebalancing` ，命令如下：

   `# ceph osd unset noout`

   `# ceph -w`

   使用 `ceph-w` 可查看集群运作输出，同步完毕后集群 health 应为 `HEALTH_OK` 状态。

4.2 恢复后检查步骤

1. 确认设备上电状态，以 IPMI 或 于设备前确认电源为开启上电状态。
2. `ping ceph monitor server` ，检查 monitor server 可以 ping 通。
3. 系统时间和校时服务器时间同步。
4. `ceph -s` 状态为 `HEALTH_OK`
5. `ceph osd tree` OSD 状态皆为 `UP`

4.3 恢复使用指令及其说明

1. `ceph -s` ：确认 ceph cluster status
2. `ceph -w` ：查看集群运作输出
3. `ceph osd tree` ：查看ceph cluster上osd排列及状态
4. `start ceph-all` ：启动 所有 ceph service
5. `start ceph-osd-all` ：启动 所有 osd service
6. `start ceph-mon-all` ：启动 所有 mon service
7. `start ceph-osd id={id}` ：启动指定 osd id service
8. `start ceph-mon id={hostname}` ：启动指定 ceph monitor host
9. `ceph osd set noout` ：ceph stopping w/out rebalancing
10. `ceph osd unset noout` ：解除ceph stopping w/out rebalancing

5. 单个Ceph节点宕机处理

在某些情况下，如服务器硬件故障，造成单台 Ceph 节点宕机无法启动，可以按照本节所示流程将该节点上的 OSD 移除集群，从而达到 Ceph 集群的恢复。

5.1 单台 Ceph 节点宕机处理步骤

1. 登陆 ceph monitor 节点，查询 ceph 状态：

   `ceph health detail`

2. 将故障节点上的所有 osd 设置成 out，该步骤会触发数据 recovery, 需要等待数据迁移完成, 同时观察虚拟机是否正常：

   `ceph osd out osd_id`

3. 从 crushmap 将 osd 移除，该步骤会触发数据 reblance，等待数据迁移完成，同时观察虚拟机是否正常：

   `ceph osd crush remove osd_name`

4. 删除 osd 的认证： `ceph auth del osd_name`
5. 删除 osd ： `ceph osd rm osd_id`

5.2 恢复后检查步骤

1. 检查 ceph 集群状态正常；
2. 检查虚拟机状态正常；
3. 楚天云人员检查虚拟机业务是否正常；
4. 检查平台服务正常：nova、cinder、glance；
5. 创建新卷正常；
6. 创建虚拟机正常。

第三部分：Ceph 进阶

这部分内容介绍了一些 Ceph 使用中的进阶技巧，主要面向二线运维人员。

1. PG 和 PGP 的区别

本篇内容来自 zphj1987 —— Ceph 中 PG 和 PGP 的区别

前言

首先来一段关于 PG 和 PGP 区别的英文解释：

> PG = Placement Group
> PGP = Placement Group for Placement purpose
> pg_num = number of placement groups mapped to an OSD
> When pg_num is increased for any pool, every PG of this pool splits into half, but they all remain mapped to their parent OSD.
> Until this time, Ceph does not start rebalancing. Now, when you increase the pgp_num value for the same pool, PGs start to migrate from the parent to some other OSD, and cluster rebalancing starts. This is how PGP plays an important role.
> By Karan Singh

以上是来自邮件列表的 `Karan Singh` 对 PG 和 PGP 的相关解释，他也是 `Learning Ceph` 和 `Ceph Cookbook` 的作者，以上的解释没有问题，我们来看下具体在集群里面如何作用。

实践

环境准备，因为是测试环境，我只准备了两台机器，每台机器 4 个 OSD，所以做了一些参数的设置，让数据尽量散列：

```
osd_crush_chooseleaf_type = 0
```

以上为修改的参数，这个是让我的环境故障域为 OSD 分组的。

创建测试需要的存储池

我们初始情况只创建一个名为 testpool 包含 6 个 PG 的存储池：

```
ceph osd pool create testpool 6 6
pool 'testpool' created
```

我们看一下默认创建完了后的 PG 分布情况：

```
ceph pg dump pgs|grep ^1|awk '{print $1,$2,$15}'
dumped pgs in format plain
1.1 0 [3,6,0]
1.0 0 [7,0,6]
1.3 0 [4,1,2]
1.2 0 [7,4,1]
1.5 0 [4,6,3]
1.4 0 [3,0,4]
```

我们写入一些对象，因为我们关心的不仅是 PG 的变动，同样关心 PG 内对象有没有移动，所以需要准备一些测试数据，这个调用原生 `rados` 接口写最方便。

```
rados -p testpool bench 20 write --no-cleanup
```

我们再来查询一次：

```
ceph pg dump pgs|grep ^1|awk '{print $1,$2,$15}'
dumped pgs in format plain
1.1 75 [3,6,0]
1.0 83 [7,0,6]
1.3 144 [4,1,2]
1.2 146 [7,4,1]
1.5 86 [4,6,3]
1.4 80 [3,0,4]
```

可以看到写入了一些数据，其中的第二列为这个 PG 当中的对象的数目，第三列为 PG 所在的 OSD。

增加 PG 测试

我们来扩大 PG 再看看：

```
ceph osd pool set testpool pg_num 12
set pool 1 pg_num to 12
```

再次查询：

```
ceph pg dump pgs|grep ^1|awk '{print $1,$2,$15}'
dumped pgs in format plain
1.1 37 [3,6,0]
1.9 38 [3,6,0]
1.0 41 [7,0,6]
1.8 42 [7,0,6]
1.3 48 [4,1,2]
1.b 48 [4,1,2]
1.7 48 [4,1,2]
1.2 48 [7,4,1]
1.6 49 [7,4,1]
1.a 49 [7,4,1]
1.5 86 [4,6,3]
1.4 80 [3,0,4]
```

可以看到上面新加上的 PG 的分布还是基于老的分布组合，并没有出现新的 OSD 组合，因为我们当前的设置是 pgp_num 为 6,那么三个 OSD 的组合的个数就是 6 个，因为当前为 12 个 PG，分布只能从 6 种组合里面挑选，所以会有重复的组合。

根据上面的分布情况，可以确定的是，增加 PG 操作会引起 PG 内部对象分裂，分裂的份数是根据新增 PG 组合重复情况来确定的，比如上面的情况：

- `1.1` 的对象分成了两份 `[3,6,0]`
- `1.3` 的对象分成了三份 `[4,1,2]`
- `1.4` 的对象没有拆分 `[3,0,4]`

结论：增加 **PG** 会引起 **PG** 内的对象分裂，也就是在 **OSD** 上创建了新的 **PG** 目录，然后进行部分对象的 **`move`** 的操作。

增加 **PGP** 测试

我们将原来的 PGP 从 6 调整到 12：

```
ceph osd pool set testpool pgp_num 12

ceph pg dump pgs|grep ^1|awk '{print $1,$2,$15}'
dumped pgs in format plain
1.a 49 [1,2,6]
1.b 48 [1,6,2]
1.1 37 [3,6,0]
1.0 41 [7,0,6]
1.3 48 [4,1,2]
1.2 48 [7,4,1]
1.5 86 [4,6,3]
1.4 80 [3,0,4]
1.7 48 [1,6,0]
1.6 49 [3,6,7]
1.9 38 [1,4,2]
1.8 42 [1,2,3]
```

可以看到 PG 里面的对象并没有发生变化，而 PG 所在的对应关系发生了变化。

我们看下与调整 PGP 前后的对比：

```
*1.1 37 [3,6,0]          1.1 37 [3,6,0]*
 1.9 38 [3,6,0]          1.9 38 [1,4,2]
*1.0 41 [7,0,6]          1.0 41 [7,0,6]*
 1.8 42 [7,0,6]          1.8 42 [1,2,3]
*1.3 48 [4,1,2]          1.3 48 [4,1,2]*
 1.b 48 [4,1,2]          1.b 48 [1,6,2]
 1.7 48 [4,1,2]          1.7 48 [1,6,0]
*1.2 48 [7,4,1]          1.2 48 [7,4,1]*
 1.6 49 [7,4,1]          1.6 49 [3,6,7]
 1.a 49 [7,4,1]          1.a 49 [1,2,6]
*1.5 86 [4,6,3]          1.5 86 [4,6,3]*
*1.4 80 [3,0,4]          1.4 80 [3,0,4]*
```

可以看到其中最原始的 6 个 PG 的分布并没有变化（标注了 `*` 号），变化的是后增加的 PG，也就是将重复的 PG 分布进行重新分布，这里并不是随机完全打散，而是根据需要去进行重分布。

结论： 调整 **PGP** 不会引起 **PG** 内的对象的分裂，但是会引起 **PG** 的分布的变动。

总结

- PG 是指定存储池存储对象的目录有多少个，PGP 是存储池 PG 的 OSD 分布组合个数。
- PG 的增加会引起 PG 内的数据进行分裂，分裂到相同的 OSD 上新生成的 PG 当中。
- PGP 的增加会引起部分 PG 的分布进行变化，但是不会引起 PG 内对象的变动。

2. Monitor 的备份和恢复

本篇内容来自 徐小胖'blog —— monitor 的增删改备

Monitor 的备份

每个 MON 的数据都是保存在数据库内的，这个数据库位于 `/var/lib/ceph/mon/$cluster-$hostname/store.db` ，这里的 `$cluster` 是集群的名字， `$hostname` 为主机名，MON 的所有数据即目录 `/var/lib/ceph/mon/$cluster-$hostname/` ，备份好这个目录之后，就可以在任一主机上恢复 MON 了。

这里参考了下 这篇文章 里面的备份方法，简单讲基本思路就是，停止一个 MON，然后将这个 MON 的数据库压缩保存到其他路径，再开启 MON。文中提到了之所以要停止 MON 是要保证 levelDB 数据库的完整性。然后可以做个定时任务一天或者一周备份一次。

另外最好把 `/etc/ceph/` 目录也备份一下。

这个备份路径最好是放到其他节点上，不要保存到本地，因为一般 MON 节点要坏就坏一台机器。

这里给出文中提到的备份方法：

```
service ceph stop mon
tar czf /var/backups/ceph-mon-backup_$(date +'%a').tar.gz /var/l
ib/ceph/mon
service ceph start mon
#for safety, copy it to other nodes
scp /var/backups/* someNode:/backup/
```

Monitor 的恢复

现在有一个 Ceph 集群，包含 3 个 monitors： ceph-1 、ceph-2 和 ceph-3 。

```
[root@ceph-1 cluster]# ceph -s
    cluster 844daf70-cdbc-4954-b6c5-f460d25072e0
      health HEALTH_OK
      monmap e2: 3 mons at {ceph-1=192.168.56.101:6789/0,ceph-2=
192.168.56.102:6789/0,ceph-3=192.168.56.103:6789/0}
            election epoch 8, quorum 0,1,2 ceph-1,ceph-2,ceph-3
      osdmap e13: 3 osds: 3 up, 3 in
        pgmap v20: 64 pgs, 1 pools, 0 bytes data, 0 objects
            101 MB used, 6125 GB / 6125 GB avail
                    64 active+clean
```

假设发生了某种故障，导致这 3 台 MON 节点全都无法启动，这时 Ceph 集群也将变得不可用。我们可以通过前面备份的数据库文件来恢复 MON。当某个集群的所有的 MON 节点都挂掉之后，我们可以将最新的备份的数据库解压到其他任意一个节点上，新建 monmap，注入，启动 MON，推送 config，重启 OSD就好了。

将 `ceph-1` 的 `/var/lib/ceph/mon/ceph-ceph-1/` 目录的文件拷贝到新节点 `ceph-4` 的 `/var/lib/ceph/mon/ceph-ceph-4/` 目录下（或者从备份路径拷贝到 `ceph-4` 节点），一定要注意目录的名称！这里 `ceph-1` 的 IP 为 `172.23.0.101` ，`ceph-4` 的 IP 为 `192.168.56.104` 。`ceph-4` 节点为一个只安装了 ceph 程序的干净节点。注意下面指令执行的节点。

```
[root@ceph-1 ~]# ip a |grep 172
    inet 172.23.0.101/24 brd 172.23.0.255 scope global enp0s8
[root@ceph-1 ~]# ping ceph-4
PING ceph-4 (192.168.56.104) 56(84) bytes of data.
64 bytes from ceph-4 (192.168.56.104): icmp_seq=1 ttl=63 time=0.
463 ms

[root@ceph-4 ~]# mkdir /var/lib/ceph/mon/ceph-ceph-4
[root@ceph-1 ~]# scp -r /var/lib/ceph/mon/ceph-ceph-1/*  ceph-4:
/var/lib/ceph/mon/ceph-ceph-4/
done
 100%    0     0.0KB/s   00:00
keyring
 100%   77     0.1KB/s   00:00
LOCK
 100%    0     0.0KB/s   00:00
LOG
 100%   21KB  20.6KB/s   00:00
161556.ldb
 100% 2098KB   2.1MB/s   00:00
......
MANIFEST-161585
 100%  709     0.7KB/s   00:00
CURRENT
 100%   16     0.0KB/s   00:00
sysvinit
 100%    0     0.0KB/s   00:00
```

同时，将 `/etc/ceph` 目录文件也拷贝到 `ceph-4` 节点，然后将 `ceph.conf` 中的 `mon_initial_members` 修改为 `ceph-4` 。

```
[root@ceph-1 ~]# scp /etc/ceph/* ceph-4:/etc/ceph/
ceph.client.admin.keyring
                  100%   63     0.1KB/s   00:00
ceph.conf
                  100%  236     0.2KB/s   00:00

[root@ceph-4 ~]# vim /etc/ceph/ceph.conf
[root@ceph-4 ~]# cat /etc/ceph/ceph.conf
[global]
fsid = 844daf70-cdbc-4954-b6c5-f460d25072e0
mon_initial_members = ceph-4
mon_host = 192.168.56.104
```

新建一个 monmap，使用原来集群的 `fsid` ，并且将 `ceph-4` 加入到这个 monmap，然后将 monmap 注入到 `ceph-4` 的 MON 数据库中，最后启动 `ceph-4` 上的 MON 进程。

```
[root@ceph-4 ~]# monmaptool --create --fsid 844daf70-cdbc-4954-b
6c5-f460d25072e0 --add ceph-4 192.168.56.104 /tmp/monmap
[root@ceph-4 ~]# ceph-mon -i ceph-4 --inject-monmap /tmp/monmap
[root@ceph-4 ~]# ceph-mon -i ceph-4
[root@ceph-4 ~]# ceph -s
    cluster 844daf70-cdbc-4954-b6c5-f460d25072e0
      health HEALTH_ERR
            64 pgs are stuck inactive for more than 300 seconds
            64 pgs degraded
            64 pgs stuck inactive
            64 pgs undersized
      monmap e6: 1 mons at {ceph-4=192.168.56.104:6789/0}
            election epoch 13, quorum 0 ceph-4
      osdmap e36: 3 osds: 1 up, 1 in
        pgmap v58: 64 pgs, 1 pools, 0 bytes data, 0 objects
            34296 kB used, 2041 GB / 2041 GB avail
                    64 undersized+degraded+peered
```

好消息是， `ceph -s` 有了正确的输出，坏消息就是 `HEALTH_ERR` 了。不过不用担心，这时候将 `ceph-4` 的 `ceph.conf` 推送到其他所有节点上，再重启 OSD 集群就可以恢复正常了。

3. 修改 Cinder/Glance 进程的最大可用 FD

本篇内容是根据生产环境中遇到的实际问题进行的总结。

背景

在生产环境中遇到这样一个问题：
下发删除卷消息无法成功删除卷，后通过 cinder 命令行命令 `cinder service-list` 查询cinder 服务状态，发现 `cinder-volume host : r2202002controller@rbd-sata` 服务状态为 `DOWN` 。

```
| cinder-volume   | r2202002controller@rbd-sata   | nova | ena
bled  |    down   |
```

该状态表明该 cinder-volume 进程已经没有正常上报心跳，处于无法处理请求的状态。

原因分析

通过现场复现问题和查看 cinder-volume 日志，发现在出问题的时间点都有删除卷的操作下发，但是有的卷一直未结束删除卷流程，直到重启 cinder-volume 进程时才恢复。

```
2016-09-23 13:42:48.176 44907 INFO cinder.volume.manager [req-01
82ed51-a4a7-44e4-bd3e-3d456610a135 ff01ec05ed4442799ebad096c1aa2
921 584c5f2fed764ec9b840319eb2cd0608 - - -] volume b38ea39e-b1f5
-4af6-a7b1-40fe1d5e80ee: deleting

2016-09-23 16:52:08.290 52145 INFO cinder.volume.manager [req-be
3fe7fc-39fe-4bc3-9a70-d5be1e7330ce - - - - -] Resuming delete on
 volume: b38ea39e-b1f5-4af6-a7b1-40fe1d5e80ee
```

怀疑在删除 RDB 卷流程有挂死问题，通过进一步查看日志，发现最后走到调用 RDB Client 删除卷时就中断了:

```
2016-09-23 13:43:27.911 44907 DEBUG cinder.volume.drivers.rbd [r
eq-0182ed51-a4a7-44e4-bd3e-3d456610a135 ff01ec05ed4442799ebad096
c1aa2921 584c5f2fed764ec9b840319eb2cd0608 - - -] deleting rbd vo
lume volume-b38ea39e-b1f5-4af6-a7b1-40fe1d5e80  delete_volume /u
sr/local/lib/python2.7/dist-packages/cinder/volume/drivers/rbd.p
y:666
```

因为该问题发生在 Ceph 集群扩容后，因此进一步怀疑和之前 Nova 挂卷后读取变慢问题一样是由于进程打开文件过多引起异常，后在实验环境中通过打开 RDB client 日志，重启 cinder-volume 进程。

```
[client]
rbd cache = true
rbd cache writethrough until flush = true
log file = /var/log/ceph/ceph.client.log
debug client = 20
debug rbd = 20
debug librbd = 20
debug objectcacher = 20
```

然后增加 cinder-volume 进程删卷时的打开文件数成功复现该问题。RDB client 日志中报 `too many open files` 异常，后在生产环境上通过复现问题和打开 RDB Client 日志观察到同样的异常，因此可以确定是由于扩容后 OSD 节点增多，RDB client 删除卷时需要和所有 OSD 建立 socket 链接，这样就会超过目前环境中 cinder-volume 进程允许打开的文件数，导致异常发生，进入挂死状态。

```
2016-09-26 20:08:48.953810 7f099566b700 -1 -- 192.168.219.2:0/43
062437 >> 192.168.219.130:6812/12006 pipe(0x7f0acdcdc090 sd=-1 :
0 s=1 pgs=0 cs=0 l=1 c=0x7f0acdcae7e0).connect couldn't created
socket (24) Too many open files
2016-09-26 20:08:48.953803 7f099556a700 -1 -- 192.168.219.2:0/43
062437 >> 192.168.219.113:6802/2740 pipe(0x7f0acdce0330 sd=-1 :0
 s=1 pgs=0 cs=0 l=1 c=0x7f0acdcb2210).connect couldn't created s
ocket (24) Too many open files
2016-09-26 20:08:48.953898 7f0995267700 -1 -- 192.168.219.2:0/43
062437 >> 192.168.219.179:6812/31845 pipe(0x7f0acdcf12f0 sd=-1 :
0 s=1 pgs=0 cs=0 l=1 c=0x7f0acdced990).connect couldn't created
socket (24) Too many open files
2016-09-26 20:08:48.953913 7f099596e700 -1 -- 192.168.219.2:0/43
062437 >> 192.168.219.169:6804/8418 pipe(0x7f0acdccd2f0 sd=-1 :0
 s=1 pgs=0 cs=0 l=1 c=0x7f0acdcc0a90).connect couldn't created s
ocket (24) Too many open files
2016-09-26 20:08:48.953932 7f0995368700 -1 -- 192.168.219.2:0/43
062437 >> 192.168.219.136:6806/23096 pipe(0x7f0acdce8f40 sd=-1 :
0 s=1 pgs=0 cs=0 l=1 c=0x7f0acdcc7cf0).connect couldn't created
socket (24) Too many open files
```

解决方案

由于问题原因是 cinder-volume 允许打开的文件数没有随着 Ceph 集群的扩容做相应的调整，因此解决方案是要调整 cinder-volume 进程的允许打开文件数，目前调整为 `65535`（根据测试，发现每个删卷请求要建立大约 1000 多个链接，因此调整为 `65535` 后可以支持并发删除约 60 个 RDB 卷，后续版本会考虑基于性能基线，进行接口并发操作数量的限制，防止无限制的并发删卷导致文件打开数过大）。

1、cinder-volume 进程被封装成了 Ubuntu 上的 upstart 任务。修改 cinder-volume 进程启动配置文件 `/etc/init/cinder-volume.conf` ，增加一行配置：

```
limit nofile 65535 65535
```

`limit` 变量用来设置任务的资源限制。

```
root@R1controller1:~# vi /etc/init/cinder-volume.conf
description "Cinder Volume"

start on (local-filesystems and net-device-up IFACE!=lo)
stop on runlevel [016]

respawn
limit nofile 65535 65535

exec su -s /bin/sh -C "exec cinder-volume --config-file /etc/cin
der/cinder.conf > /dev/null 2>&1" root
```

2、重启 cinder-volume 进程，并用 `cat proc/pid/limits` 查询设置是否生效。

```
root@R1controller1:~# service cinder-volume restart
cinder-volume stop/waiting
cinder-volume start/running, process 6298

root@R1controller1:~# cat /etc/6298/limits | grep open
Max open files       65535                65535                fil
es
```

3、检查 cinder-volume 服务是否为 `UP` 状态。

```
root@R1controller1:~# cinder service-list
+------------------+---------------------------+------+---------
+-------+---------------------------+-----------------+
|      Binary      |            Host           | Zone |  Status
| State |         Updated_at         | Disabled Reason |
+------------------+---------------------------+------+---------
+-------+---------------------------+-----------------+
|  cinder-backup   |      OPS-controller1      | nova | enabled
|   up  | 2016-09-07T20:38:30.000000 |       None      |
|  cinder-backup   |      OPS-controller2      | nova | enabled
|   up  | 2016-09-07T20:38:26.000000 |       None      |
| cinder-scheduler |      OPS-controller1      | nova | enabled
|   up  | 2016-09-07T20:38:29.000000 |       None      |
| cinder-scheduler |      OPS-controller2      | nova | enabled
|   up  | 2016-09-07T20:38:29.000000 |       None      |
|  cinder-volume   | OPS-controller1@netapp_fc | nova | enabled
|   up  | 2016-09-07T20:38:32.000000 |       None      |
|  cinder-volume   |  OPS-controller1@rbd-sas  | nova | enabled
|   up  | 2016-09-07T20:38:33.000000 |       None      |
|  cinder-volume   |  OPS-controller1@rbd-ssd  | nova | enabled
|   up  | 2016-09-07T20:38:33.000000 |       None      |
|  cinder-volume   |  OPS-controller2@rbd-sas  | nova | enabled
|   up  | 2016-09-07T20:38:25.000000 |       None      |
|  cinder-volume   |  OPS-controller2@rbd-ssd  | nova | enabled
|   up  | 2016-09-07T20:38:27.000000 |       None      |
+------------------+---------------------------+------+---------
+-------+---------------------------+-----------------+
```

4、重新进行删除卷操作，卷可以被正常删除。

5、因为 Glance 后端也对接的是 Ceph 集群，为防止 Glance 也出现该问题，建议也修改 Glance 进程的启动配置文件 `/etc/init/glance-api.conf` ，增加 `limit nofile 65535 65535` 配置行。

6、重启 glance-api 进程。

```
service glance-api restart
```

7、重启完成后上传测试镜像，测试镜像可以正常上传和删除。

4. 更换 OSD Journal

本篇中部分内容来自 zphj1987 —— 如何替换 Ceph 的 Journal

Ceph 在一块单独的磁盘上部署 OSD 的时候，是默认把 journal 和 OSD 放在同一块磁盘的不同分区上。有时候，我们可能需要把 OSD 的 journal 分区从一个磁盘替换到另一个磁盘上去。那么应该怎样替换 Ceph 的 journal 分区呢？

有两种方法来修改 Ceph 的 journal：

- 创建一个 journal 分区，在上面创建一个新的 journal。
- 转移已经存在的 journal 分区到新的分区上，这个适合整盘替换。

> Ceph 的 journal 是基于事务的日志，所以正确的下刷 journal 数据，然后重新创建 journal 并不会引起数据丢失，因为在下刷 journal 的数据的时候，osd 是停止的，一旦数据下刷后，这个 journal 是不会再有新的脏数据进来的。

第一种方法

1、首先给 Ceph 集群设置 `noout` 标志。

```
root@mon:~# ceph osd set noout
set noout
```

2、假设我们现在想要替换 osd.0 的 journal。首先查看 osd.0 当前的 journal 位置，当前使用的是 `/dev/sdb2` 分区。

```
root@mon:~# ceph-disk list | grep osd.0
  /dev/sdb1 ceph data, active, cluster ceph, osd.0, journal /dev
/sdb2

root@mon:~# ll /var/lib/ceph/osd/ceph-0/journal
lrwxrwxrwx 1 root root 58 May 24 15:06 /var/lib/ceph/osd/ceph-0/
journal -> /dev/disk/by-partuuid/8e95b09d-ffa9-4163-b24c-b780200
22797
root@mon:~# ls -l /dev/disk/by-partuuid/
total 0
lrwxrwxrwx 1 root root 10 Nov  8 09:21 39e9ad34-d7aa-4dec-865e-0
8952aa8aab5 -> ../../sdc1
lrwxrwxrwx 1 root root 10 Nov  8 09:21 8e95b09d-ffa9-4163-b24c-b
78020022797 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Nov  8 09:21 aaeca5fa-456a-4f45-8a8b-9
de0c2642f44 -> ../../sdc2
lrwxrwxrwx 1 root root 10 Nov  8 09:21 d30a6d4a-6da4-4a81-a9e5-4
bc69ebeec8f -> ../../sdb1
```

3、停止 osd.0 进程。

```
stop ceph-osd id=0
```

4、下刷 journal 到 osd，使用 `-i` 指定需要替换 journal 的 osd 的编号。

```
root@mon:~# ceph-osd -i 0 --flush-journal
SG_IO: bad/missing sense data, sb[]:  70 00 05 00 00 00 00 0a 00
 00 00 00 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
SG_IO: bad/missing sense data, sb[]:  70 00 05 00 00 00 00 0a 00
 00 00 00 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
2016-11-08 13:17:58.355025 7f8351a72800 -1 flushed journal /var/
lib/ceph/osd/ceph-0/journal for object store /var/lib/ceph/osd/c
eph-0
```

5、删除旧的 journal 。

```
root@mon:~# ll /var/lib/ceph/osd/ceph-0/journal
lrwxrwxrwx 1 root root 58 May 24 15:06 /var/lib/ceph/osd/ceph-0/
journal -> /dev/disk/by-partuuid/8e95b09d-ffa9-4163-b24c-b780200
22797
root@mon:~# rm -rf /var/lib/ceph/osd/ceph-0/journal
```

6、下面用 `/dev/sdc2` 分区重建 osd.0 的 journal 。查看 `/dev/sdc2` 的 `uuid` ：

```
root@mon:~# ls -l /dev/disk/by-partuuid/
total 0
lrwxrwxrwx 1 root root 10 Nov  8 09:21 39e9ad34-d7aa-4dec-865e-0
8952aa8aab5 -> ../../sdc1
lrwxrwxrwx 1 root root 10 Nov  8 13:17 8e95b09d-ffa9-4163-b24c-b
78020022797 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Nov  8 09:21 aaeca5fa-456a-4f45-8a8b-9
de0c2642f44 -> ../../sdc2
lrwxrwxrwx 1 root root 10 Nov  8 09:21 d30a6d4a-6da4-4a81-a9e5-4
bc69ebeec8f -> ../../sdb1
```

新的 journal 的 uuid 的路径为 `/dev/disk/by-partuuid/aaeca5fa-456a-4f45-8a8b-9de0c2642f44` 。

7、新建 journal 的链接和 journal_uuid 文件：

```
root@mon:~# ln -s /dev/disk/by-partuuid/aaeca5fa-456a-4f45-8a8b-
9de0c2642f44 /var/lib/ceph/osd/ceph-0/journal
root@mon:~# echo aaeca5fa-456a-4f45-8a8b-9de0c2642f44 > /var/lib
/ceph/osd/ceph-0/journal_uuid
```

8、给 osd.0 创建 journal，使用 `-i` 指定 osd 的编号 。

```
root@mon:~# ceph-osd -i 0 --mkjournal
SG_IO: bad/missing sense data, sb[]:  70 00 05 00 00 00 00 0a 00
 00 00 00 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
2016-11-08 13:29:36.115461 7f64ec851800 -1 created new journal /
var/lib/ceph/osd/ceph-0/journal for object store /var/lib/ceph/o
sd/ceph-0
```

9、查看新 journal 。

```
root@mon:~# ceph-disk list | grep osd.0
  /dev/sdb1 ceph data, active, cluster ceph, osd.0, journal /dev
/sdc2
```

10、启动 osd.0 。

```
start ceph-osd id=0
```

11、去除 noout 的标记。

```
ceph osd unset noout
```

12、检查集群的状态。

```
root@mon:~# ceph -s
cluster 614e77b4-c997-490a-a3f9-e89aa0274da3
 health HEALTH_OK
 monmap e5: 1 mons at {osd1=10.95.2.43:6789/0}
        election epoch 796, quorum 0 osd1
 osdmap e1067: 3 osds: 3 up, 3 in
        flags sortbitwise
  pgmap v309733: 384 pgs, 6 pools, 1148 MB data, 311 objects
        3597 MB used, 73162 MB / 76759 MB avail
             384 active+clean

root@mon:~# ceph osd tree
ID WEIGHT  TYPE NAME                                    UP/DOWN
 REWEIGHT PRIMARY-AFFINITY
-4 0.05997 root default

-1 0.01999     host mon

 0 0.01999         osd.0                                     up
  1.00000          1.00000
-2 0.01999     host osd0

 1 0.01999         osd.1                                     up
  1.00000          1.00000
-3 0.01999     host osd1

 2 0.01999         osd.2                                     up
  1.00000          1.00000
```

第二种方法

这个属于备份和转移分区表的方法。

1、首先按方法一中的第 1 ~ 4 步，设置 `noout` 标志，停进程，下刷 journal。

2、备份需要替换 journal 的分区表。

```
root@lab8106 ~# sgdisk --backup=/tmp/backup_journal_sdd /dev/sdd
```

3、 还原分区表。

```
root@lab8106 ~# sgdisk --load-backup=/tmp/backup_journal_sde /de
v/sde
root@lab8106 ~# parted -s /dev/sde print
```

新的 journal 磁盘现在跟老的 journal 的磁盘的分区表一样的了。这意味着新的分区的 UUID 和老的相同的。如果选择的是这种备份还原的方法，那么 journal 的那个软连接是不需要进行修改的，因为两个磁盘的 uuid 是一样的，所以需要注意将老的磁盘拔掉或者清理掉分区，以免冲突。

4、重建 journal 。

```
root@lab8106 ~# ceph-osd -i 0 --mkjournal
```

5、启动进程。

```
root@lab8106 ~# start ceph-osd id=0
```

6、去除 `noout` 的标记。

```
root@lab8106 ~# ceph osd unset noout
```

5. 清空 OSD 的分区表后如何恢复

本篇内容来自 zphj1987 —— 不小心清空了 Ceph 的 OSD 的分区表如何恢复

假设不小心对 Ceph OSD 执行了 `ceph-deploy disk zap` 这个操作，那么该 OSD 对应磁盘的分区表就丢失了。本文讲述了在这种情况下如何进行恢复。

破坏环境

我们现在有一个正常的集群，假设用的是默认的分区的方式，我们先来看看默认的分区方式是怎样的。

1、查看默认的分区方式。

```
root@mon:~# ceph-disk  list
...
/dev/sdb :
  /dev/sdb1 ceph data, active, cluster ceph, osd.0, journal /dev
/sdb2
  /dev/sdb2 ceph journal, for /dev/sdb1
...
```

2、查看分区情况

```
root@mon:~# parted -s /dev/sdb print
Model: SEAGATE ST3300657SS (scsi)
Disk /dev/sdb: 300GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name          Flags
  2      1049kB  1074MB  1073MB               ceph journal
  1      1075MB  300GB   299GB   xfs           ceph data
```

3、破坏 `/dev/sdb` 的分区表，该磁盘对应的是 `osd.0` 。

```
root@mon:~/ceph# ceph-deploy disk zap mon:/dev/sdb
[ceph_deploy.conf][DEBUG ] found configuration file at: /root/.c
ephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked (1.5.34): /usr/bin/ceph-deploy
 disk zap mon:/dev/sdb
...
[mon][DEBUG ] Warning: The kernel is still using the old partiti
on table.
[mon][DEBUG ] The new table will be used at the next reboot.
[mon][DEBUG ] GPT data structures destroyed! You may now partiti
on the disk using fdisk or
[mon][DEBUG ] other utilities.
...
```

即使这个 osd 在被使用，还是被破坏了，这里假设上面的就是一个误操作，我们看下带来了哪些变化：

```
root@mon:~/ceph# ll /var/lib/ceph/osd/ceph-0/journal
lrwxrwxrwx 1 root root 58 Sep 24 00:02 /var/lib/ceph/osd/ceph-0/
journal -> /dev/disk/by-partuuid/bd81471d-13ff-44ce-8a33-92a8df9
e8eee
```

如果你用命令行看，就可以看到上面的链接已经变红了，分区没有了：

```
root@mon:~/ceph# ceph-disk  list
/dev/sdb :
 /dev/sdb1 other, xfs, mounted on /var/lib/ceph/osd/ceph-0
 /dev/sdb2 other
```

已经跟上面有变化了，没有 ceph 的相关分区信息了：

```
root@mon:~/ceph# parted -s /dev/sdb print
Model: SEAGATE ST3300657SS (scsi)
Disk /dev/sdb: 300GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start  End  Size  File system  Name  Flags
```

分区表完全没有信息了，到这我们可以确定分区表完全没了，如果现在重启将会发生什么？重启以后这个磁盘就是一个裸盘，没有分区的裸盘，所以此时千万不能重启！

恢复环境

首先一个办法就是当这个 OSD 坏了，然后直接按照删除节点，添加节点的方法去处理，这个应该是最主流、最通用的处理办法，但是这个方法在生产环境当中引发的数据迁移还是非常大的。我们尝试做恢复，这就是本篇主要讲的东西。

1、首先设置 `noout` 标志。

```
root@mon:~/ceph# ceph osd set noout
```

2、停止 OSD 。

```
root@mon:~/ceph# stop ceph-osd id=0
```

现在的 OSD 还是有进程的，所以需要停止掉再做处理。

3、查看其他 OSD 的分区信息（这里要求磁盘一致）。

```
root@mon:~/ceph# parted -s /dev/sdc unit s print
Model: SEAGATE ST3300657SS (scsi)
Disk /dev/sdc: 585937500s
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start     End         Size        File system  Name
     Flags
 2      2048s     2097152s    2095105s                 ceph jour
nal
 1      2099200s  585937466s  583838267s  xfs          ceph data
```

记住上面的数值， `print` 的时候是加了 `unit s` 这个是要精确的值的，下面的步骤会用到的这些数值。

4、进行分区表的恢复。

```
root@mon:~/ceph# parted -s /dev/sdb mkpart ceph_data 2099200s 58
5937466s
root@mon:~/ceph# parted -s /dev/sdb mkpart ceph_journal 2048s 20
97152s
```

5、再次检查 /dev/sdb 的分区表。

```
root@mon:~/ceph# parted -s /dev/sdb print
Model: SEAGATE ST3300657SS (scsi)
Disk /dev/sdb: 300GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name          Flags
 2      1049kB  1074MB  1073MB               ceph_journal
 1      1075MB  300GB   299GB   xfs          ceph_data
```

可以看到，分区表已经回来了。

6、重新挂载分区。

```
root@mon:~/ceph# umount /var/lib/ceph/osd/ceph-0
root@mon:~/ceph# partprobe
root@mon:~/ceph# mount /dev/sdb1 /var/lib/ceph/osd/ceph-0
```

7、删除旧的 journal ，重建 osd.0 的 journal。

```
root@mon:~/ceph# rm -rf /var/lib/ceph/osd/ceph-0/journal

root@mon:~/ceph# ceph-osd -i 0 --osd-journal=/dev/sdb2 --mkjourn
al
SG_IO: bad/missing sense data, sb[]:  70 00 05 00 00 00 00 0a 00
 00 00 00 20 00 01 cf 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
2016-09-24 00:36:06.595992 7f9d0afbc880 -1 created new journal /
dev/sdb2 for object store /var/lib/ceph/osd/ceph-0
root@mon:~/ceph# ln -s /dev/sdb2 /var/lib/ceph/osd/ceph-0/journa
l
root@mon:~/ceph# ll /var/lib/ceph/osd/ceph-0/journal
lrwxrwxrwx 1 root root 9 Sep 24 00:37 journal -> /dev/sdb2
```

注意上面的操作 `--osd-journal=/dev/sdb2` 这个地方，此处写成 `/dev/sdb2` 是便于识别，这个地方在实际操作中要写上 `/dev/sdb2` 的 uuid 的路径。

```
root@mon:~/ceph# ll /dev/disk/by-partuuid/03fc6039-ad80-4b8d-86e
c-aeee14fb3bb6
lrwxrwxrwx 1 root root 10 Sep 24 00:33 /dev/disk/by-partuuid/03f
c6039-ad80-4b8d-86ec-aeee14fb3bb6 -> ../../sdb2
```

也就是这个链接的一串内容，这是为了防止盘符串了的情况下无法找到 journal 的问题。

8、启动 OSD 。

```
root@mon:~/ceph# start ceph-osd id=0
```

检查下，到这 osd.0 就成功地恢复了。

6. PG 卡在 active + remapped 状态

问题现象

有时，我们的 Ceph 集群可能会出现 PG 长时间卡在 `active + remapped` 的状态。

```
root@ceph1:~# ceph -s
    cluster 5ccdcb2d-961d-4dcb-a9ed-e8034c56cf71
      health HEALTH_WARN 88 pgs stuck unclean
      monmap e2: 1 mons at {ceph1=192.168.56.102:6789/0}, electi
on epoch 1, quorum 0 ceph1
      osdmap e71: 4 osds: 4 up, 3 in
        pgmap v442: 256 pgs, 4 pools, 285 MB data, 8 objects
            690 MB used, 14636 MB / 15326 MB avail
                    88 active+remapped
                  168 active+clean
```

产生问题的原因

出现这种情况，一般是做了 osd 的 `reweight` 操作引起的，这是因为一般在做 `reweight` 的操作的时候，根据算法，这个上面的 pg 是会尽量分布在这个主机上的，而 `crush reweight` 不变的情况下，去修改 osd 的 `reweight` 的时候，可能算法上会出现无法映射的问题。

如何解决

1、直接做 `ceph osd crush reweigh` 的调整即可避免这个问题，这个 straw 算法里面还是有点小问题的，在调整某个因子的时候会引起整个因子的变动。

2、从 FIREFLY (CRUSH_TUNABLES3) 开始 CRUSH 里面增加了一个参数：

```
chooseleaf_vary_r
```

是否递归的进行 chooseleaf 尝试，如果非 0 ，就递归的进行，这个基于 parent 已经做了多少次尝试。默认值是 0 ，但是常常找不到合适的 mapping 。在计算成本和正确性上来看最优值是 1 。对于已经有大量数据的集群来说，从 0 调整为 1 将会有大量数值的迁移，调整为 4 或者 5 的话，将会找到一个更有效的映射，可以减少数据的移动。

查看当前的值：

```
root@ceph1:~# ceph osd crush show-tunables |grep chooseleaf_vary
_r
    "chooseleaf_vary_r": 0,
```

修改 `chooseleaf_vary_r` 的值。

Hammer 版本下这个参数默认为：

> ```
> tunable chooseleaf_vary_r 0
> ```

修改 Crush Map 的方法请参考本手册第一部分 9. 管理 Crushmap 。

或者，直接修改 `crush tunables` 的值为 `optimal` 。

```
ceph osd crush tunables optimal
```

7. 查看 RBD 镜像的位置

有时，我们需要查看某个 RBD 镜像的对象都存放在哪些 PG 中，这些 PG 又分布在哪些 OSD 上。可以利用下面的 shell 脚本来实现快速查看 RBD 镜像的位置。

```
#!/bin/bash

# USAGE:./rbd-loc <pool> <image>

if [ -z ${1} ] || [ -z ${2} ];
then
    echo "USAGE: ./rbd-loc <pool> <image>"
exit 1
fi

rbd_prefix=$(rbd -p ${1} info ${2} | grep block_name_prefix | aw
k '{print $2}')
for i in $(rados -p ${1} ls | grep ${rbd_prefix})
do
    ceph osd map ${1} ${i}
done
```

执行的效果如下所示：

```
root@mon:~# rbd ls -p images
fc5b017d-fc74-4a59-80bb-a5a76e26dd4e

root@mon:~# ./rbd-loc.sh images fc5b017d-fc74-4a59-80bb-a5a76e26
dd4e
osdmap e1078 pool 'images' (9) object 'rbd_data.1349f035c101d9.0
000000000000001' -> pg 9.99b52d94 (9.14) -> up ([1,2,0], p1) act
ing ([1,2,0], p1)
osdmap e1078 pool 'images' (9) object 'rbd_data.1349f035c101d9.0
000000000000002' -> pg 9.40973ca2 (9.22) -> up ([0,2,1], p0) act
ing ([0,2,1], p0)
osdmap e1078 pool 'images' (9) object 'rbd_data.1349f035c101d9.0
000000000000003' -> pg 9.86758b2c (9.2c) -> up ([1,2,0], p1) act
ing ([1,2,0], p1)
osdmap e1078 pool 'images' (9) object 'rbd_data.1349f035c101d9.0
000000000000004' -> pg 9.3c8e78f6 (9.36) -> up ([0,1,2], p0) act
ing ([0,1,2], p0)
osdmap e1078 pool 'images' (9) object 'rbd_data.1349f035c101d9.0
000000000000000' -> pg 9.ffc971ff (9.3f) -> up ([0,2,1], p0) act
ing ([0,2,1], p0)
```

该测试环境只有 3 个 host， 每个 host 上 1 个 OSD ，3 副本设置。

8. 查看 RBD 镜像的实际大小

本篇内容来自 zphj1987 —— 如何统计 Ceph 的 RBD 真实使用容量

Ceph 的 rbd 一直有个问题就是无法清楚的知道这个分配的空间里面到底使用了多少，使用 `rbd info` 命令查询出来的容量是预分配的总容量而非实际使用容量。在 Jewel 版中提供了一个新的接口去查询，对于老版本来说可能同样有这个需求，本篇将详细介绍如何解决这个问题。

目前已知的有三种查询方法：

1. 使用 `rbd du` 查询（Jewel 版才支持）
2. 使用 `rbd diff`
3. 根据对象统计的方法进行统计

方法一：使用 rbd du 查询

此命令在 Jewel 版中可用。

```
root@mon:~# rbd du rbd/mysql-img
NAME PROVISIONED USED
test       52.8047M    0
```

不过需要注意，执行此命令要求开启 rbd image 的如下属性：

```
layering, exclusive-lock, object-map, fast-diff
```

具体使用可参考 这篇文章 。

方法二：使用 rbd diff

```
root@mon:~# rbd diff rbd/mysql-img | awk '{ SUM += $2 } END { pr
int SUM/1024/1024 " MB" }'
52.8047 MB
```

方法三：根据对象统计的方法进行统计

在集群非常大的时候，再按上面的方法一个个查询，需要花很长的时间，并且需要时不时的跟集群进行交互。方法三是把统计数据一次获取下来，然后进行数据的统计分析，从而获取结果，获取的粒度是以存储池为基准的。

拿到所有对象的信息：

```
for obj in `rados -p rbd ls`;do rados -p rbd stat $obj >> obj.tx
t;done
```

这个获取的时间长短是根据对象的多少来的，如果担心出问题，可以换个终端查看进度：

```
tail -f obj.txt
```

获取 RBD 的镜像列表：

```
rbd -p rbd ls
img2
mysql-img
volume-e5376906-7a95-48bb-a1c6-bb694b4f4813.backup.base
```

获取 RBD 的镜像的 prefix ：

```
root@mon:~# for a in `rbd -p rbd ls`;do echo $a ;rbd -p rbd info
 $a|grep prefix |awk '{print $2}' ;done
img2
rb.0.f4730.2ae8944a
mysql-img
rb.0.f4652.2ae8944a
volume-e5376906-7a95-48bb-a1c6-bb694b4f4813.backup.base
rbd_data.23a53c28fb938f
```

获取指定RBD镜像的大小：

```
root@mon:~# cat obj.txt |grep rb.0.f4652.2ae8944a |awk  '{ SUM +
= $6 } END { print SUM/1024/1024 " MB" }'
52.8047 MB
```

将上面的汇总，使用脚本一次查询出指定存储池中所有镜像的大小：

```
#!/bin/bash
# USAGE:./get_used.sh <poolname>

objfile=obj.txt
Poolname=${1}
echo "In the pool ${Poolname}":

for obj in `rados -p $Poolname ls`
do
    rados -p $Poolname stat $obj >> $objfile
done

for image in `rbd -p $Poolname ls`
do
    Imagename=$image
    Prefix=`rbd  -p $Poolname info $image|grep prefix |awk '{pri
nt $2}'`
    Used=`cat $objfile |grep $Prefix|awk '{ SUM += $6 } END { pr
int SUM/1024/1024 " MB" }'`
    echo $Imagename $Prefix
    echo Used: $Used
done
```

执行的效果如下：

```
root@mon:~# ./get_used.sh rbd
In the pool rbd:
img2 rb.0.f4730.2ae8944a
Used: 3076 MB
mysql-img rb.0.f4652.2ae8944a
Used: 158.414 MB
volume-e5376906-7a95-48bb-a1c6-bb694b4f4813.backup.base rbd_data
.23a53c28fb938f
Used: 96 MB
```

注意这里只统计了 image 里面的真实容量，如果是用了链接 clone 的，存在容量复用的问题，需要自己看是否需要统计那一部分的对象，方法同上。

9. 统计 OSD 上 PG 的数量

我们可以通过一个 Python 脚本，统计出每个 OSD 上分布了多少个 PG ，以此判断集群的数据分布是否均衡。

```
#!/usr/bin/env  python
import sys
import os
import json
cmd = '''
ceph pg dump | awk ' /^pg_stat/ { col=1; while($col!="up") {col+
+}; col++ } /^[0-9a-f]+\.[0-9a-f]+/ {print $1,$col}'
'''
body = os.popen(cmd).read()
SUM = {}
for line in  body.split('\n'):
       if not line.strip():
        continue
       SUM[line.split()[0]] = json.loads(line.split()[1])
pool = set()
for  key in  SUM:
      pool.add(key.split('.')[0])
mapping = {}
for number in pool:
      for k,v in SUM.items():
        if k.split('.')[0] == number:
               if number in mapping:
                   mapping[number] += v
               else:
                   mapping[number] = v
MSG = """%(pool)-6s: %(pools)s | SUM
%(line)s
%(dy)s
%(line)s
%(sun)-6s: %(end)s |"""
pools = " ".join(['%(a)-6s' % {"a": x} for x in sorted(list(mapp
```

```
ing))])
line = len(pools) + 20
MA = {}
OSD = []
for p in mapping:
    osd = sorted(list(set(mapping[p])))
    OSD += osd
    count = sum([mapping[p].count(x) for x in osd])
    osds = {}
    for x in osd:
        osds[x] = mapping[p].count(x)
    MA[p] = {"osd": osds, "count": count}
MA = sorted(MA.items(), key=lambda x:x[0])
OSD = sorted(list(set(OSD)))
DY = ""
for osd in OSD:
    count = sum([x[1]["osd"].get(osd,0) for x in MA])
    w = ["%(x)-6s" % {"x": x[1]["osd"].get(osd,0)} for x in MA]
    #print w
    w.append("| %(x)-6s" % {"x": count})
    DY += 'osd.%(osd)-3s %(osds)s\n' % {"osd": osd, "osds": " ".
join(w)}
SUM = " ".join(["%(x)-6s" % {"x": x[1]["count"]} for x in MA])
msg = {"pool": "pool", "pools": pools, "line": "-" * line, "dy":
 DY, "end": SUM, "sun": "SUM"}
print MSG % msg
```

执行效果如下：

```
root@OPS-ceph1:~# ./cal_pg_per_osd.py
dumped all in format plain
pool  : 0      1      2      3      4      5      7      | SUM
----------------------------------------------------------------
----
osd.0   2      9      11     10     5      6      1      | 44

osd.1   1      7      8      10     8      9      3      | 46

osd.2   2      11     7      6      6      8      4      | 44
```

```
osd.3    1     11    7     9     7     4     1     | 40
osd.4    2     12    12    12    11    13    0     | 62
osd.5    2     10    10    5     9     11    0     | 47
osd.6    3     56    47    49    38    43    16    | 252
osd.7    6     36    48    45    55    42    10    | 242
osd.8    4     41    42    37    35    49    15    | 223
osd.9    6     42    52    41    55    36    12    | 244
osd.10   10    36    47    51    39    43    15    | 241
osd.11   6     56    47    44    41    46    12    | 252
osd.12   6     40    45    45    51    46    11    | 244
osd.13   6     42    40    56    46    44    9     | 243
osd.14   5     44    41    49    48    52    7     | 246
osd.15   4     50    42    38    49    38    12    | 233
osd.16   3     9     5     11    8     8     2     | 46
osd.17   2     10    13    4     7     10    2     | 48
osd.18   0     12    10    10    10    9     6     | 57
osd.19   0     13    8     4     9     15    2     | 51
osd.20   1     11    7     9     9     15    2     | 54
osd.21   0     7     12    10    6     12    1     | 48
osd.22   4     39    45    30    43    46    12    | 219
```

```
osd.23  6       44      38      44      38      41      6       | 217

osd.24  7       37      50      49      49      36      8       | 236

osd.25  10      28      38      44      40      32      11      | 203

osd.26  1       44      33      47      47      37      13      | 222

osd.27  3       47      38      42      44      55      14      | 243

osd.28  4       47      47      34      33      48      11      | 224

osd.29  3       40      32      34      40      51      10      | 210

osd.30  8       39      46      47      55      35      10      | 240

osd.31  5       48      48      45      41      36      11      | 234

osd.32  5       46      48      53      42      48      7       | 249


----------------------------------------------------------------
----
SUM   : 128     1024    1024    1024    1024    1024    256     |
```

10. 查看使用 RBD 镜像的客户端

有时候删除 rbd image 会提示当前 rbd image 正在使用中，无法删除：

```
rbd rm foo
2016-11-09 20:16:14.018332 7f81877a07c0 -1 librbd: image has wat
chers - not removing
Removing image: 0% complete...failed.
rbd: error: image still has watchers
This means the image is still open or the client using it crashe
d. Try again after closing/unmapping it or waiting 30s for the c
rashed client to timeout.
```

所以希望能查看到底是谁在使用 rbd image。

对于 rbd image 的使用，目前主要有两种形式：内核模块 map 后再 mount ；通过 libvirt 等供给虚拟机使用。都是利用 `rados listwatchers` 去查看，只是两种形式下需要读取的文件不一样。

内核模块 map

查看方法如下：

```
# 查看 rbd pool 中的 image
root@ceph1:~# rbd ls
foo
ltest
test

# 查看 foo 的使用者
root@ceph1:~# rados -p rbd listwatchers foo.rbd
watcher=10.202.0.82:0/1760414582 client.1332795 cookie=1

# 去对应主机上检查
root@ceph2:~#rbd showmapped |grep foo
0  rbd  foo   -    /dev/rbd0
```

虚拟机通过 libvirt 使用 rbd image

查看方法如下：

```
# 查看该虚拟机卷的信息
root@ceph1:~# rbd info volumes/volume-ee0c4077-a607-4bc9-a8cf-e8
93837361f3
rbd image 'volume-ee0c4077-a607-4bc9-a8cf-e893837361f3':
        size 1024 MB in 256 objects
        order 22 (4096 kB objects)
        block_name_prefix: rbd_data.13c3277850f538
        format: 2
        features: layering, striping
        flags:
        parent: images/3601459f-060b-460f-b73b-db74237d922e@snap
        overlap: 40162 kB
        stripe unit: 4096 kB
        stripe count: 1

# 查看该 image 的使用者
root@ceph1:~# rados -p volumes listwatchers rbd_header.13c327785
0f538
watcher=10.202.0.21:0/1043073 client.1298745 cookie=140256077850
496
```

我们登录控制节点，查看对应的 cinder 卷：

```
root@controller:~# cinder list | grep ee0c4077-a607-4bc9-a8cf-e8
93837361f3
| ee0c4077-a607-4bc9-a8cf-e893837361f3 | in-use |      |  1   |
     -      |   true   | 96ee1aad-af27-4c9d-968f-291dbb2766a1 |
```

该卷挂载在 ID 为 `96ee1aad-af27-4c9d-968f-291dbb2766a1` 的虚拟机上。通过 `nova show` 命令验证该虚拟机是否在物理机 10.202.0.21 (computer21)上：

```
root@controller:~# nova show 96ee1aad-af27-4c9d-968f-291dbb2766a
1
+--------------------------------------+------------------------
----------------------------------------------------------+
| Property                             | Value
                                                          |
+--------------------------------------+------------------------
----------------------------------------------------------+
| OS-DCF:diskConfig                    | AUTO
                                                          |
| OS-EXT-AZ:availability_zone          | az_ip
                                                          |
| OS-EXT-SRV-ATTR:host                 | computer21
                                                          |
| OS-EXT-SRV-ATTR:hostname             | byp-volume
                                                          |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | computer21
                                                          |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000989
                                                          |
| OS-EXT-SRV-ATTR:kernel_id            |
                                                          |
| OS-EXT-SRV-ATTR:launch_index         | 0
                                                          |
| OS-EXT-SRV-ATTR:ramdisk_id           |
                                                          |
| OS-EXT-SRV-ATTR:reservation_id       | r-hwiyx15c
                                                          |
| OS-EXT-SRV-ATTR:root_device_name     | /dev/vda
```

```
                                                         |
| OS-EXT-SRV-ATTR:user_data             | -
                                                         |
| OS-EXT-STS:power_state                | 1
                                                         |
| OS-EXT-STS:task_state                 | -
                                                         |
| OS-EXT-STS:vm_state                   | active
                                                         |
| OS-SRV-USG:launched_at                | 2016-11-09T08:19:41.000
000                                                      |
| OS-SRV-USG:terminated_at              | -
                                                         |
| accessIPv4                            |
                                                         |
| accessIPv6                            |
                                                         |
| blue-net network                      | 192.168.50.27
                                                         |
| config_drive                          |
                                                         |
| created                               | 2016-11-09T07:53:59Z
                                                         |
| description                           | byp-volume
                                                         |
| flavor                                | m1.small (2)
                                                         |
| hostId                                | 5104ee1a0538048d6ef80b1
4563a0cbac461f86523c5c81f5d18069e                        |
| host_status                           | UP
                                                         |
| id                                    | 96ee1aad-af27-4c9d-968f
-291dbb2766a1                                            |
| image                                 | Attempt to boot from vo
lume - no image supplied                                 |
| key_name                              | octavia_ssh_key
                                                         |
| locked                                | False
                                                         |
| metadata                              | {}
```

```
                                                        |
| name                                  | byp-volume
                                                        |
| os-extended-volumes:volumes_attached | [{"id": "ee0c4077-a607-
4bc9-a8cf-e893837361f3", "delete_on_termination": true}] |
| progress                              | 0
                                                        |
| security_groups                       | default
                                                        |
| status                                | ACTIVE
                                                        |
| tenant_id                             | f21a9c86d7114bf99c711f4
874d80474                                               |
| updated                               | 2016-11-09T08:19:41Z
                                                        |
| user_id                               | 142d8663efce464c89811c6
3e45bd82e                                               |
| zq-test network                       | 192.168.6.6
                                                        |
+--------------------------------------+------------------------
----------------------------------------------------------+
```